Fundamenta Informaticae XX (2019) 1–32

DOI 10.3233/FI-2016-0000

IOS Press

From Box Algebra to Interval Temporal Logic

Hanna Klaudel
IBISC, University of Evry, Université Paris-Saclay
91025, Evry, France
hanna.klaudel@ibisc.univ-evry.fr

Maciej Koutny
School of Computing, Newcastle University
Newcastle upon Tyne, NE4 5TG, United Kingdom
maciej.koutny@ncl.ac.uk

Zhenhua Duan
ICTT and ISN Laboratory, Xidian University
Xi'an, 710071, P.R. China
zhhduan@mail.xidian.edu.cn

Ben Moszkowski
School of Computing, Newcastle University
Newcastle upon Tyne, NE4 5TG, United Kingdom
benmos63@gmail.com

Abstract. In this paper, we further develop a recently introduced semantic link between temporal logics and Petri nets. We focus on two specific formalisms, Interval Temporal Logic (ITL) and Box Algebra (BA), which are closely related by their compositional approach to constructing system descriptions. The overall goal of our investigation is to translate Petri nets into behaviourally equivalent logical formulas. As a result, the analysis of system properties can be carried out using either of the two formalisms, exploiting their respective strengths and powerful tool support.

The contribution of this paper is twofold. First, we extend the existing translation from BA to ITL, by removing restrictions concerning the way control flow of concurrent system is modelled, and by allowing a fully general synchronisation operator. Second, we strengthen the notion of equivalence between a Petri net and the corresponding logical formula by proving such an equivalence at the level of transition-based executions of Petri nets rather than just by looking at their labels. We also show that the complexity of the proposed translation compares favourably with the complexity of the translation from BA expressions to Petri nets.

Keywords: Interval Temporal Logic, Petri nets, box algebra, composition, semantics, general synchronisation, step sequence, equivalence

1. Introduction

Temporal logics [1, 2, 3, 4] and Petri nets [5, 6] are generally regarded as fundamentally different approaches to the specification and analysis of concurrent systems. The former allow one to specify both the system designs and correctness requirements within a single logical framework, and the verification of correctness can be done by checking the satisfaction of formulas, or model checking. In contrast, Petri nets are an automata-inspired model with semantics based on actions and local states which allows one to capture causal relationships in systems' behaviour. As a result, verification of behavioural properties can be carried out using invariant techniques [7], based on the graph structure of nets, or model checking techniques, based on partial order semantics [8, 9].

To establish a semantic link between logics and Petri nets, we focused in [10] on two specific formalisms, *Interval Temporal Logic* (ITL) [11, 12] and *Box Algebra* (BA) [13], which are closely related by the their compositional approaches to constructing system descriptions. In particular, in both ITL and BA the control flow of a system is specified by means of commonly used programming operators, such as sequence, choice, parallelism, and iteration. The synchronisation between concurrently executed subsystems is, however, achieved in different ways and therefore needs to be suitably handled.

Box Algebra [13], and its precursor Petri Box Calculus [14], provided a generic process-algebraic syntax together with a compositional translation to a class of Petri nets called boxes. This generic algebra has several concrete incarnations, including CCS [15] and TCSP [16]. BA can also readily yield a compositional semantics for an imperative concurrent programming language [17]. It has also been extended to handle, e.g., stochastic aspects of concurrent systems [18].

In contrast to point-based temporal logics (e.g., [2, 3]), where computations are described in terms of successive system states, interval temporal logics aim at capturing the evolution of a system referring to its behaviour over time intervals. As pointed out in [19], the concept of time intervals is both naturally appealing and intuitive, and has been of significant interest to several disciplines, including Philosophy, Linguistics, Artificial Intelligence, and Computer Science. For example, interval temporal logics allow one to easily capture concepts relating to action duration or shared-variable concurrency (e.g., [20]). The preface [4] of a journal special issue on interval temporal logics identified, in particular, two categories of temporal logics formalisms for intervals: (i) modal formalisms such as HS-logics [21] and CDT [22] with interval-based variables and capable of expressing Allen's relations [23] (recent developments concerning model checking HS-logics include [24, 25, 26, 27, 28]); and (ii) Interval Temporal Logic (ITL) of [29, 30, 12] together with its variations and adaptations which use state-based variables and the sequential composition 'chop' operator to a significant extent. In this paper we focus on ITL which has influenced several research groups and projects, as well as IEEE standard 1647 ([31]). It is not unusual for researchers to develop their own variants of ITL, often with quite different names. The Duration Calculus [32], a prominent real-time variant of ITL, is the best example. A recent instance of this is Multi-Lane Spatial Logic [33], which is intended for modeling road traffic (e.g, for computer-based assistance of car drivers). ITL lends itself to execution, and can support abstract specifications and concrete implementations in the same notation, with refinement mappings between. Moreover, timings of executions can be easily derived by considering interval lengths.

1.1. About this paper

In [10], we proved the correctness of a translation from a submodel of BA to semantically equivalent ITL formulas. The submodel we considered disallowed the nesting of the parallel composition operator. Moreover, synchronisation was binary. In the paper [34], we demonstrated how to extend the submodel of [10] to handle data variables.

In this paper, which is a full and extended version of a conference publication [35], we provide a syntax-driven translation for the core BA [13] syntax comprising parallel composition, sequence, choice, synchronisation, and iteration but without considering data variables (incorporating these using the technique introduced in [34] would be straightforward but, at the same time, would obscure the key constructs introduced in the present paper). Crucially, we relax the syntactical constraints preventing the use of the parallel composition outside the topmost level of process expressions. Moreover, we consider a fully general synchronisation operator. Finally, we strengthen the notion of equivalence between a BA net and the corresponding ITL formula by proving such an equivalence at the level of transition-based executions of Petri nets, rather than just by taking their labels. It is worth noting that for the purposes of this paper, i.e., the translation of the basic BA, we only need to use propositional ITL (PITL).

To formalise a semantic link between BA and ITL, for every logical formula we introduce a step sequence semantics which records variables changing their values at each computational step. This allows us to compare its behaviour with the step sequences of the corresponding Petri net, and to conclude their full equivalence in the main result (Theorem 4.3). In essence, the latter states that any property which can be captured within the step sequence model can be analysed using either of the two equivalent representations, i.e., a Petri net or a logical formula.

1.2. Paper organisation

The paper is organised as follows. In the next section, we recall the basic notions of Box Algebra. We start with the definition of box expressions, which are then used to compositionally construct box nets. We also recall a number of results demonstrating how the execution semantics of a composite box can be derived from the execution semantics of its components. In Section 3, we similarly recall the relevant fragment of Interval Temporal Logic, and present some basic semantic properties of ITL formulas. Section 4 is central to the whole paper as it contains a formal translation from box expressions to ITL formulas as well as the proof of a semantic equivalence between box expressions and the corresponding formulas. The following section presents examples of this translation, and Section 6 briefly discusses future work.

1.3. An introductory example

In order to outline the main ideas behind the proposed solution, we will now consider a small fragment of an application involving two concurrent processes, a server and a client, following a simple interactive protocol where: (i) the client process sends a query to a database server, awaits the answer, and then carries out a local update; and (ii) the server process non-deterministically chooses between

handling a client request or performing its own local update:

Client	Server
⋮	⋮
(send.req ; receive.ans) ; local.upd	if
⋮	case 1 : receive.req ; send.ans
	case 2 : local.upd
	fi
	⋮

From the above fragment of pseudo-code we can extract in a straightforward way the following Box Algebra representation, where ; and □ respectively denote sequential composition and non-deterministic choice:

$$Client \;=\; (s.req \,;\, r.ans) \,;\, l.upd \qquad\qquad Server \;=\; (r.req \,;\, s.ans) \,\square\, l.upd.$$

The box semantics of the above expressions is obtained compositionally, by first associating basic boxes (single-transition Petri nets) with atomic box expressions, and then by applying Box Algebra compositions rules. In general, Petri nets are directed graphs with two kind of nodes: *places (local states)* represented by circles, and *transitions (actions)* represented by squares or rectangles. The arcs connect nodes of different types and indicate direct relationships between places and transitions. In the box nets, additionally, circles labelled with e are entry places, circles labelled with i are internal places, and circles labelled with x are exit places, reflecting their intended role in net composition.

The successive construction stages of the box semantics for $Server$

$$N_{Server} = N_{(r.req \,;\, s.ans) \,\square\, l.upd} = (N_{r.req} \,;\, N_{s.ans}) \,\square\, N_{l.upd}$$

are shown in Figure 1. First, two basic boxes $N_{r.req}$ and $N_{s.ans}$ (the two Petri nets in Figure 1(a)) are composed sequentially by gluing the exit (or final) place of $N_{r.req}$ with the entry (or initial) place of $N_{s.ans}$. The resulting box $N_{r.req \,;\, s.ans}$ (the Petri net in Figure 1(b)) is then composed with the basic box $N_{l.upd}$ by applying the non-deterministic choice operation, which glues together their entry places as well as their exit places (the final Petri net is shown in Figure 1(c)).

The net N_{Server} we have just constructed provides the graph-theoretic (static) representation of the $Server$ process. To obtain its dynamic (behavioural) representation, we need in addition to specify the initial marking (or state) of $Server$, from which one can explore all possible execution paths by following the standard transition firing rules. In the case of a box net, this is simply done by inserting a single token in each of its entry places (in this case, just one place). The initial marking can then be used to investigate the dynamic behaviour of $Server$, which is basically composed of two execution paths, both ending at a final marking in which the only exit place contains a token. One such path executes $r.req$ followed by $s.ans$, and the other executes a single transition $l.upd$.

The states of the $Server$ process are all the markings reachable from the initial one, and possible executions are represented by sequences of executed transitions (or sets of transitions, called steps).

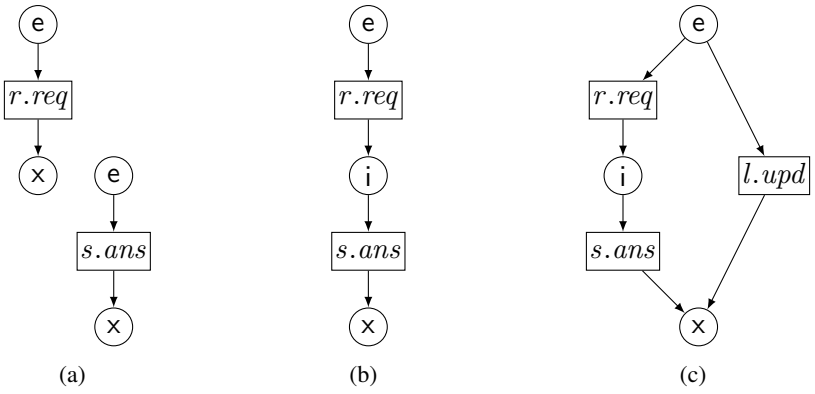


Figure 1. Compositional derivation of the box $N_{Server} = N_{(r.req\,;\,s.ans)\,\square\,l.upd}$: (a) basic boxes representing $r.req$ and $s.ans$; (b) sequential composition representing $r.req\,;\,s.ans$; and (c) non-deterministic choice representing $Server$. Note that the entry and exit places are respectively labelled by e and x; moreover, i identifies the internal places. All transitions are labelled by the corresponding atomic expressions occurring in the $Server$ process.

In the ITL context, however, a state is a mapping which assigns values to a set of (Boolean) variables, and possible executions are represented by sequences of such assignments, called intervals. Therefore, we need a method to bridge the gap between these two different ways of representing the states and executions of concurrent systems. The solution we adopt and develop in this paper is to associate with each transition t of N_{Server} a distinct Boolean variable, also denoted by t, and then to represent each firing of transition t by flipping the value of variable t; otherwise, the value of variable t is kept unchanged. With this idea in mind, the box expression $Server$ can be captured by the following ITL formula:

$$\mathsf{itl}(Server) \quad = \quad (\mathsf{fs}(r.req \mid s.ans, l.upd)\,;\mathsf{fs}(s.ans \mid r.req, l.upd)) \quad \vee \quad \mathsf{fs}(l.upd \mid r.req, s.ans).$$

where ; is the ITL "chop" operator[1] (sequential composition) and a formula of the form $\mathsf{fs}(v \mid w, u)$ means that the values of the two variables w and u are kept unchanged while the value of v can be flipped once within the interval over which the formula is evaluated (if v is never flipped, the interval must be infinite). The execution path of N_{Server} in which transition $r.req$ is followed by $s.ans$ can then be matched by a corresponding interval satisfying $\mathsf{itl}(Server)$ in which the variables $r.req$ and $s.ans$ flip their values, as shown below:

executed transitions	$r.req$	$s.ans$	
$r.req$	0	1	1
$s.ans$	0	0	1
$l.upd$	0	0	0

[1] Subsequently adopted, e.g., by the Duration Calculus [32].

Having seen how $Server = (r.req \,;\, s.ans) \,\Box\, l.upd$ can be represented by an ITL formula, one might attempt to do exactly the same for a concurrent composition of both processes:

$$\begin{array}{l} \big((\mathsf{fs}(s.req \mid \ldots) \,;\, \mathsf{fs}(r.ans \mid \ldots)) \quad ; \quad \mathsf{fs}(l.upd \mid \ldots)\big) \\ \wedge \\ \big((\mathsf{fs}(r.req \mid \ldots) \,;\, \mathsf{fs}(s.ans \mid \ldots)) \quad \vee \quad \mathsf{fs}(l.upd \mid \ldots)\big) \end{array}$$

Such a naive translation exhibits two serious problems. The first is that it does not provide any means to synchronise the communication actions between the two processes. We address this issue by identifying some of the variables in different processes, such as $s.req$ and $r.req$. The other problem is quite an opposite one, namely, the formula specifies that two different local updates must happen simultaneously as they are represented by the same ITL Boolean variable $l.upd$. We address this problem by introducing two different variables which are allowed to flip their values independently of each other. Before illustrating how both solutions work, we will complete the presentation of the net model for the example system.

The whole synchronised system comprising a client and a server is captured by the following box expression, where $\|$ is the parallel composition operator, and sco (for scoping) is an operator specifying all interprocess synchronisations [13]. For example, $s.req, r.req \mapsto req$ means that the simultaneously executed actions $s.req$ and $r.req$ can be synchronised, and replaced by a single new action req, while $l.upd \mapsto upd$ indicates that each action $l.upd$ is simply maintained, after being renamed. Here is the resulting system expressed in Box Algebra, where the upper line represents server and the lower receiver:

$$System = \left(\begin{array}{c} (s.req \,;\, r.ans) \,;\, l.upd \\ \| \\ (r.req \,;\, s.ans) \,\Box\, l.upd \end{array} \right) \mathsf{sco} \left\{ \begin{array}{rcl} s.req, r.req & \mapsto & req \\ s.ans, r.ans & \mapsto & ans \\ l.upd & \mapsto & upd \end{array} \right\}$$

The box N_{System} generated from the above expression is depicted in Figure 2. *A crucial point is that during its generation each action (and then the corresponding transition) is made unique by annotating it with the path(s) reflecting its position in the parse tree of the System expression.* (Note that the construction stages shown in Figure 1 would also be amended to include appropriate annotations.) For example, $r.req_{\|_R \,\Box_L \,;_L}$ means that $r.req$ is reachable in the parse tree through the following path: from the root, labelled with $\|$, go to its right child (R) corresponding to the second operand, labelled with $\Box$, and from there visit its left child (L) corresponding to the first operand, labelled with $;$ and, in turn, from here visit the left child (L) corresponding to the first operand, which is the node labelled with $r.req$.

Thanks to annotating the individual actions with unique syntax paths, one can avoid unwanted synchronisations of independent actions, and then obtain an ITL translation of the whole synchronised system expression:

$$\begin{array}{rcl} \mathsf{itl}(System) & = & \big((\mathsf{fs}(\xi \mid \alpha, \gamma) \,;\, \mathsf{fs}(\alpha \mid \xi, \gamma)) \quad ; \quad \mathsf{fs}(\gamma \mid \xi, \alpha)\big) \\ & & \wedge \\ & & \big((\mathsf{fs}(\xi \mid \alpha, \delta) \,;\, \mathsf{fs}(\alpha \mid \xi, \delta)) \quad \vee \quad \mathsf{fs}(\delta \mid \xi, \alpha)\big) \end{array}$$

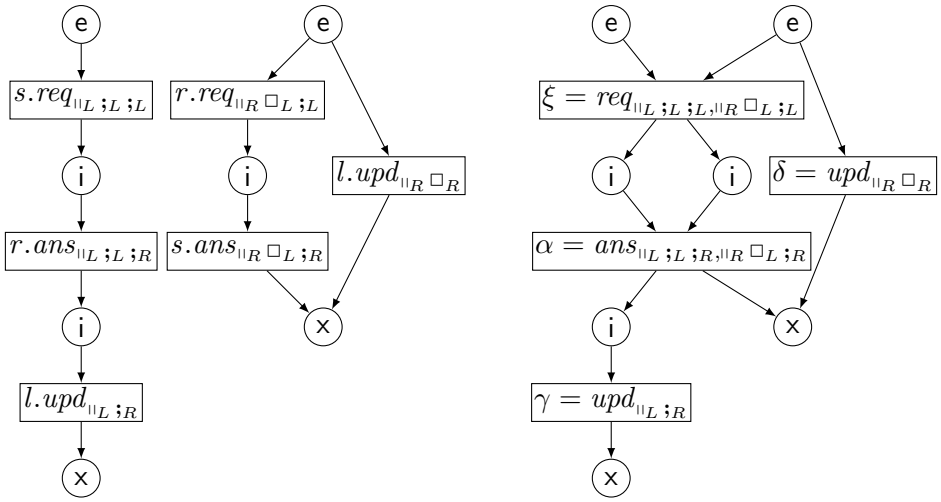


Figure 2. Boxes of $Client \parallel Server$ (a), and $System$ (b).

where $\xi = req_{||_L\,;L\,;L,||_R\,\square_L\,;L}$, $\alpha = ans_{||_L\,;L\,;R,||_R\,\square_L\,;R}$, $\gamma = upd_{||_L\,;R}$, and $\delta = upd_{||_R\,\square_R}$. In this paper, we show that the executions of the box shown in Figure 2 are consistent with the intervals satisfying the ITL formula $\mathsf{itl}(System)$ as, for example, in the following execution scenario:

executed transitions	ξ	α	γ	
ξ	0	1	1	1
α	0	0	1	1
γ	0	0	0	1
δ	0	0	0	0

1.4. Notation

The concatenation operator for sequences will be denoted by "$\cdot$". For a finite sequence δ and a set of sequences Δ, $\delta\cdot\Delta = \{\delta\cdot\delta' \mid \delta' \in \Delta\}$, and for a set of finite sequences Δ and a set of sequences Δ', $\Delta\cdot\Delta' = \{\delta\cdot\delta' \mid \delta \in \Delta \wedge \delta' \in \Delta'\}$. The k-th element of a sequence δ will be denoted by $\delta_{(k)}$ and its length by $|\delta|$ ($|\delta| = \omega$ if δ is infinite). We will use the following notations involving sequences of empty sets: $\varnothing^* = \{\epsilon, \varnothing, \varnothing\varnothing, \dots\}$ is the infinite set of all finite sequences of empty sets, and $\varnothing^\omega = \{\varnothing\varnothing\dots\}$ is the set containing a single infinite sequence of empty sets.

2. Box Algebra

Let $\mathcal{A}$ be a finite set of atomic actions. A synchronisation relation ρ is a finite set of tuples of actions $(a_1, \ldots, a_n, a)$ with $n \geq 1$. Intuitively, $a_1, \ldots, a_n$ represent n concurrent actions which can be synchronised to yield a single composite action with the label a. To reflect this intuition, we will often denote $(a_1, \ldots, a_n, a)$ by $a_1 \ldots a_n \mapsto a$.

The syntax given below defines two kinds of box expressions, namely, non-synchronised expressions E capturing the control flow in a concurrent system, and synchronised expressions F (below a is an action and ρ a synchronisation relation):

$$\begin{array}{lcl} E & ::= & \mathsf{stop} \mid a \mid E\,;E \mid E \,\Box\, E \mid E \parallel E \mid [\![E \circledast E \circledast E]\!] \\ F & ::= & E \,\mathsf{sco}\, \rho \end{array}$$

The intuition behind the above syntax is that: (i) stop denotes a blocked process; (ii) a denotes a process which can execute an action $a \in \mathcal{A}$ and terminate; (iii) $E\,;E'$ denotes sequential composition; (iv) $E \,\Box\, E'$ denotes non-deterministic choice composition; (v) $E \parallel E'$ denotes parallel composition; (vi) $[\![E \circledast E' \circledast E'']\!]$ denotes a loop with an initial part E, iterated part E', and terminal part E''; and (vii) $E \,\mathsf{sco}\, \rho$ denotes scoping, which first creates all the synchronisations specified by ρ and then deletes all the actions of E.

2.1. Box nets

The semantics of box expressions is given through a mapping into Petri nets called boxes.

A *box* is a tuple $N = (P, T, F, \ell)$, where: (i) P and T are disjoint finite sets of respectively *places* (representing local states) and *transitions* (representing actions); (ii) $F \subseteq (P \times T) \cup (T \times P)$ is a *flow relation*; and (iii) ℓ is a *labelling function* for places and transitions such that $\ell(p) \in \{\mathsf{e}, \mathsf{i}, \mathsf{x}\}$ associates an entry, internal, or exit status with every place p, and an atomic action $\ell(t) \in \mathcal{A}$, with every transition t.

The sets of entry, internal and exit places of N are given respectively by $N^{\mathsf{e}} = \ell^{-1}(\mathsf{e})$, $N^{\mathsf{i}} = \ell^{-1}(\mathsf{i})$, and $N^{\mathsf{x}} = \ell^{-1}(\mathsf{x})$. Moreover, we set $N^{\mathsf{ei}} = N^{\mathsf{e}} \cup N^{\mathsf{i}}$ and $N^{\mathsf{ix}} = N^{\mathsf{i}} \cup N^{\mathsf{x}}$, and will use N^{P}, N^{T}, and N^{F} to respectively denote the places, transitions, and the flow relation (arcs) of N. We also adopt the standard rules about representing nets as directed graphs.

2.2. Semantics of box nets

The global states of a box N are called *markings*, each marking being a mapping M assigning a non-negative integer to every place in N^{P}. The default initial (or entry) and final (or exit) markings of N, denoted respectively by M_N^{e} and M_N^{x}, are defined, for every $p \in N^{\mathsf{P}}$, as follows:

$$M_N^{\mathsf{e}}(p) = \begin{cases} 1 & \text{if } p \in N^{\mathsf{e}} \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad M_N^{\mathsf{x}}(p) = \begin{cases} 1 & \text{if } p \in N^{\mathsf{x}} \\ 0 & \text{otherwise.} \end{cases}$$

The change of a marking of N results from a simultaneous execution of a (possibly empty) set of transitions, called a *step*. Formally, a step of N is any set of transitions $U \subseteq N^{\mathsf{T}}$. It is enabled at a

marking M if, for every place $p \in N^{\mathsf{P}}$:

$$M(p) \geq |\{t \in U \mid (p,t) \in F\}|.$$

We denote this by $M[U\rangle$. An enabled step U can be executed leading to a marking M' given, for every place $p \in N^{\mathsf{P}}$, by:

$$M'(p) = M(p) - |\{t \in U \mid (p,t) \in F\}| + |\{t \in U \mid (t,p) \in F\}|.$$

We denote this by $M[U\rangle M'$.

The semantics of N is given through its infinite step sequences starting from the default initial marking M_N^{e}. In addition, we single out a set of finite step sequences which lead from the default initial marking M_N^{e} to the default final marking M_N^{x}. Intuitively, each such step sequence will be interpreted as a successfully terminating execution of N.

A *step sequence* of N is an infinite sequence $\theta = U_1 U_2 \ldots$ of steps such that there exist markings $M_1, M_2, \ldots$ of N satisfying

$$M_N^{\mathsf{e}}[U_1\rangle M_1 \;\; M_1[U_2\rangle M_2 \;\; M_2[U_3\rangle M_3 \;\; \ldots$$

Moreover, we define a terminating step sequence of N as a finite sequence $\theta = U_1 \ldots U_m$ $(m \geq 0)$ of steps such that there exist markings $M_1, \ldots, M_{m-1}$ of N satisfying

$$M_N^{\mathsf{e}}[U_1\rangle M_1 \;\; M_1[U_2\rangle M_2 \;\; M_2[U_3\rangle M_3 \;\; \ldots \;\; M_{m-1}[U_m\rangle M_N^{\mathsf{x}}.$$

The sets of (infinite) step sequences and terminating step sequences of N are respectively denoted by $\mathsf{infsts}(N)$ and $\mathsf{finsts}(N)$.[2]

2.3. Composite boxes

The above definition of a box net is too general for our purposes, as we are interested in nets derived compositionally from box expressions. The labelling of places will provide the necessary device for composing boxes along the entry and exit interfaces, i.e., the sets of entry places N^{e} and exit places N^{x}.

The nets we are going to construct will have a very specific form of places and transitions, making it easier to establish connections between boxes and ITL formulas. Intuitively, we will use the syntax of a non-synchronised box expression E to construct concrete places and transitions of the corresponding box N by embedding paths from the root of the parse tree of E in the definitions of places and transitions. In what follows, finite sequences in the set

$$\Pi = \{ ;_L, ;_R, \Box_L, \Box_R, \|_L, \|_R, \circledast_L, \circledast_M, \circledast_R \}^*$$

will be called syntax paths. Note that symbols appearing in syntax paths correspond to the arguments of operators used in box expressions (with 'L' indicating the left operand, $\circledast_M$ the middle operand,

[2]Note that a deadlock corresponds to a step sequence $\theta \in \mathsf{infsts}(N)$ such that it can be decomposed as $\theta = \theta' \cdot \varnothing\varnothing \ldots$ and $\{\theta'' \mid \theta' \cdot \theta'' \in \mathsf{infsts}(N)\} = \varnothing^\omega$.

etc). For two syntax paths, π_1 and π_2, we use $\pi_1|\pi_2$ to denote that $\{\pi_1, \pi_2\} = \{\pi\cdot{}_{||_L}\cdot\pi', \pi\cdot{}_{||_R}\cdot\pi''\}$, for some π, π', and π''. Intuitively, two actions of a non-synchronised box expression are concurrent iff their positions π_1 and π_2 in the syntax (or parse) tree are such that $\pi_1|\pi_2$.

The form of each place in composite boxes will be p_Z, where $p \in \{\mathsf{e}, \mathsf{i}, \mathsf{x}\}$ and $Z \subseteq \Pi\cdot\{\mathsf{e}, \mathsf{x}\}$, while each transition will be of the form a_W, where $a \in \mathcal{A}$ and $W \subseteq \Pi$. Moreover, for brevity, the sets Z and W will be written as comma-separated lists without brackets.

For a syntax path $\pi \in \Pi$ and a transition a_W, we denote $\pi\cdot a_W = a_{\pi\cdot W}$. This prefix notation extends in the usual way to sets of transitions and sequences of sets of transitions as well as (sets of) places.

The specific form of places and transitions, together with the systematic way in which boxes are manipulated below, will mean that for a compositional box $N = (P, T, F, \ell)$ it will be the case that, for all $p_Z \in P$ and $a_W \in T$, $\ell(p_Z) = p$ and $\ell(a_W) = a$, as well as:

$$\begin{array}{rcl} (p_Z, a_W) \in F & \iff & \exists \pi \in W : \pi\cdot\mathsf{e} \in Z \\ (a_W, p_Z) \in F & \iff & \exists \pi \in W : \pi\cdot\mathsf{x} \in Z. \end{array} \tag{1}$$

Thus both the flow relation and labelling function are implicit, and we will represent such a box simply by $N = (P, T)$.

We will now present a systematic way of constructing composite boxes from box expressions, i.e., for each constant expression we define a box and, for each operator used in box expressions, a corresponding operator on boxes.

Constants With the blocking expression stop and a single-action expression $a \in \mathcal{A}$, we respectively associate the following boxes:

$$N_{\mathsf{stop}} = (\{\mathsf{e}_\mathsf{e}, \mathsf{x}_\mathsf{x}\}, \varnothing) \quad \text{and} \quad N_a = (\{\mathsf{e}_\mathsf{e}, \mathsf{x}_\mathsf{x}\}, \{a_\epsilon\}). \tag{2}$$

Their diagrams are depicted in Figure 3 with labels shown inside the nodes. N_{stop} consists of one entry place, one exit place, and nothing else. N_a contains, in addition, a single transition. The way it is connected with the two places is determined by the formula (1). For example, $(\mathsf{e}_\mathsf{e}, a_\epsilon) \in F$ since $\pi = \epsilon$ is an annotation of a_ϵ and $\pi\cdot\mathsf{e} = \mathsf{e}$ is an annotation of e_e.

Sequence $N\,;K$ combines the exit interface of N with the entry interface of K. The entry interface of the resulting box is that of N, and the exit interface is that of K. For an example see the diagram of $N_a\,;N_b$ in Figure 3.

$$N\,;K \;=\; \big(\; P_L \,\cup\, P_R \,\cup\, \mathcal{X}\;,\; T_L \,\cup\, T_R \;\big)$$

$$\text{where} \left\{ \begin{array}{lcllcl} T_L & = & {}_{;L}\cdot N^{\mathsf{T}} & T_R & = & {}_{;R}\cdot K^{\mathsf{T}} \\ P_L & = & {}_{;L}\cdot N^{\mathsf{ei}} & P_R & = & {}_{;R}\cdot K^{\mathsf{ix}} \\ \mathcal{X} & = & \{\mathsf{i}_{{}_{;L}\cdot Z \,\cup\, {}_{;R}\cdot W} \mid \mathsf{x}_Z \in N^{\mathsf{x}} \wedge \mathsf{e}_W \in K^{\mathsf{e}}\}. & & & \end{array} \right.$$

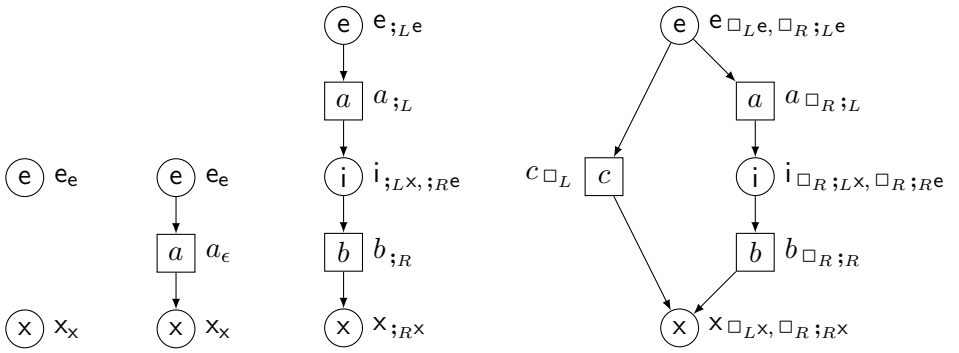


Figure 3. From the left to right: diagrams of N_{stop} and N_a and $N_a \,;\, N_b$ and $N_c \,\Box\, (N_a \,;\, N_b)$. The last two boxes correspond to the box expressions $a \,;\, b$ and $c \,\Box\, (a \,;\, b)$.

Non-deterministic choice $N \,\Box\, K$ combines together the entry interfaces of the two boxes creating a new entry interface, as well as their exit interfaces creating a new exit interface. For an example, see the diagram of $N_c \,\Box\, (N_a \,;\, N_b)$ in Figure 3.

$$N \,\Box\, K \;=\; \big(\, P_L \,\cup\, P_R \,\cup\, \mathcal{X} \,\cup\, \mathcal{Y}\,,\; T_L \,\cup\, T_R \,\big)$$

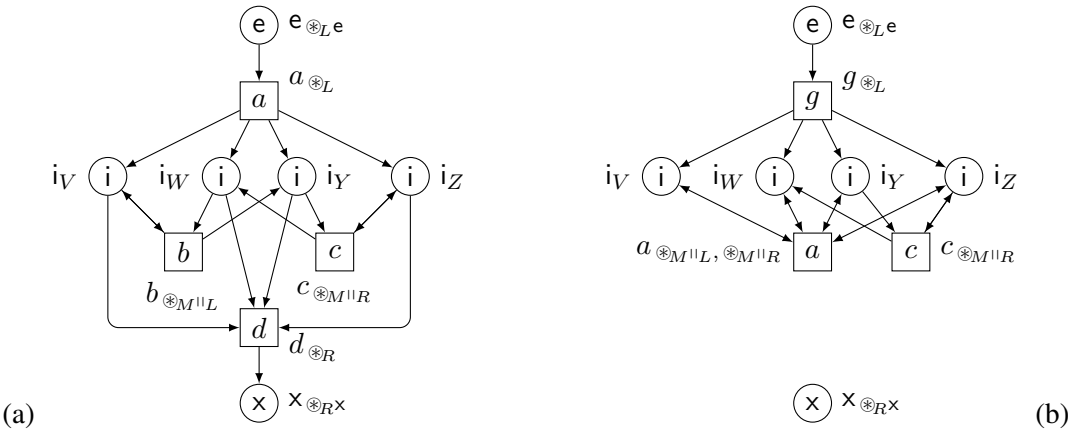
$$\text{where} \left\{ \begin{array}{lcllcl} T_L & = & \Box_L \cdot N^{\mathsf{T}} & \qquad T_R & = & \Box_R \cdot K^{\mathsf{T}} \\ P_L & = & \Box_L \cdot N^{\mathsf{i}} & \qquad P_R & = & \Box_R \cdot K^{\mathsf{i}} \\ \mathcal{X} & = & \{\mathsf{e}_{\Box_L \cdot Z \,\cup\, \Box_R \cdot W} \mid \mathsf{e}_Z \in N^{\mathsf{e}} \wedge \mathsf{e}_W \in K^{\mathsf{e}}\} & & & \\ \mathcal{Y} & = & \{\mathsf{x}_{\Box_L \cdot Z \,\cup\, \Box_R \cdot W} \mid \mathsf{x}_Z \in N^{\mathsf{x}} \wedge \mathsf{x}_W \in K^{\mathsf{x}}\}. & & & \end{array} \right.$$

Parallelism $N \parallel K$ simply puts the boxes N and K next to each other. The new entry (resp., exit) interface is the union of the entry (resp., exit) interfaces of the composed boxes.

$$N \parallel K \;=\; \big(\, P_L \,\cup\, P_R\,,\; T_L \,\cup\, T_R \,\big)$$

$$\text{where} \left\{ \begin{array}{lcllcl} T_L & = & \shortparallel_L \cdot N^{\mathsf{T}} & \qquad T_R & = & \shortparallel_R \cdot K^{\mathsf{T}} \\ P_L & = & \shortparallel_L \cdot N^{\mathsf{P}} & \qquad P_R & = & \shortparallel_R \cdot K^{\mathsf{P}} \end{array} \right.$$

Iteration $[\![N \circledast K \circledast J]\!]$ combines the exit interfaces of N and K with the entry interfaces of K and J, respectively. For an example see the diagram of $[\![N_a \circledast (N_b \parallel N_c) \circledast N_d]\!]$ in Figure 4(a). The



$$\mathsf{i}_V = \mathsf{i}_{\circledast_L \mathsf{x}, \circledast_{M||_L} \mathsf{e}, \circledast_{M||_L} \mathsf{x}, \circledast_R \mathsf{e}} \qquad \mathsf{i}_W = \mathsf{i}_{\circledast_L \mathsf{x}, \circledast_{M||_L} \mathsf{e}, \circledast_{M||_R} \mathsf{x}, \circledast_R \mathsf{e}}$$
$$\mathsf{i}_Y = \mathsf{i}_{\circledast_L \mathsf{x}, \circledast_{M||_R} \mathsf{e}, \circledast_{M||_L} \mathsf{x}, \circledast_R \mathsf{e}} \qquad \mathsf{i}_Z = \mathsf{i}_{\circledast_L \mathsf{x}, \circledast_{M||_R} \mathsf{e}, \circledast_{M||_R} \mathsf{x}, \circledast_R \mathsf{e}}.$$

Figure 4. Diagrams of two boxes involving parallel composition and iteration: $[\![N_a \circledast (N_b \parallel N_c) \circledast N_d]\!]$ (a), and $([\![N_a \circledast (N_b \parallel N_c) \circledast N_d]\!])\,\mathsf{sco}\,\{a \mapsto g, bc \mapsto a, c \mapsto c\}$ (b). The four internal places are defined below the diagrams.

new entry interface is that of N, and the exit interface is that of J.

$$[\![N \circledast K \circledast J]\!] = \left(P_L \cup P_M \cup P_R \cup \mathcal{X} \,,\, T_L \cup T_M \cup T_R \right)$$

$$\text{where} \left\{ \begin{array}{llllll} T_L = \circledast_L \cdot N^{\mathsf{T}} & T_M = \circledast_M \cdot K^{\mathsf{T}} & T_R = \circledast_R \cdot J^{\mathsf{T}} \\ P_L = \circledast_L \cdot N^{\mathsf{ei}} & P_M = \circledast_M \cdot K^{\mathsf{i}} & P_R = \circledast_R \cdot J^{\mathsf{ix}} \\ \mathcal{X} = \{\mathsf{i}_{\circledast_L \cdot Z \cup \circledast_M \cdot W \cup \circledast_M \cdot V \cup \circledast_R \cdot Y} \mid \\ \qquad \mathsf{x}_Z \in N^{\mathsf{x}} \wedge \mathsf{e}_W \in K^{\mathsf{e}} \wedge \mathsf{x}_V \in K^{\mathsf{x}} \wedge \mathsf{e}_Y \in J^{\mathsf{e}}\}. \end{array} \right.$$

At this point it is possible to formulate a useful result which holds for all boxes which can be constructed using the rules defined so far (clearly, any net constructed in this way is a box).

Proposition 2.1. Let N be any net constructed from boxes corresponding to the constant box expressions as well as the operators for sequence, choice, parallelism, and iteration. Then, the set of steps appearing in the step sequences of N is included in the following set:

$$N^{\mathsf{psteps}} = \left\{ \{a^1_{\pi_1}, \ldots, a^n_{\pi_n}\} \subseteq N^{\mathsf{T}} \mid \forall i < j : \pi_i | \pi_j \right\}.$$

Moreover, if the net N_{stop} defined in (2) is not used in the construction, then N^{psteps} is exactly the set of steps appearing in the step sequences of N.

Proof:
Follows by structural induction from the general results proved, e.g., in [13] for Box Algebra models.
□

The last result states that the valid steps of N (i.e., those occurring in the step sequences of N) do not contain conflicting or causally dependent transitions, i.e., transitions whose syntactic paths π and π' do not satisfy $\pi|\pi'$. Henceforth we will call N^{psteps} the set of potential steps of N. Moreover, for every synchronisation relation ρ, we define:

$$\rho_N = \{(U, a_{\pi_1,\ldots,\pi_n}) \mid U = \{a^1_{\pi_1}, \ldots, a^n_{\pi_n}\} \in N^{\mathsf{psteps}} \wedge (a^1, \ldots, a^n, a) \in \rho\}.$$

Each pair $(U, a_W) \in \rho_N$ comprises a set of transitions U which in principle might occur simultaneously in a valid step of N and, under the synchronisation specified by ρ in $N \,\mathsf{sco}\, \rho$, would give rise to a compound transition a_W. For example, if we take the box N in Figure 4(a) with the synchronisation relation $\rho = \{a \mapsto g, bc \mapsto a, c \mapsto c\}$, then we have:

$$\begin{array}{rcl} N^{\mathsf{psteps}} & = & \{\varnothing, \{a_{\circledast_L}\}, \{b_{\circledast_{M|\!|_L}}\}, \{c_{\circledast_{M|\!|_R}}\}, \{d_{\circledast_R}\}, \{b_{\circledast_{M|\!|_L}}, c_{\circledast_{M|\!|_R}}\}\} \\ \rho_N & = & \{(\{a_{\circledast_L}\}, g_{\circledast_L}), (\{c_{\circledast_{M|\!|_R}}\}, c_{\circledast_{M|\!|_R}}), (\{b_{\circledast_{M|\!|_L}}, c_{\circledast_{M|\!|_R}}\}, a_{\circledast_{M|\!|_L}, \circledast_{M|\!|_R}})\}\,. \end{array}$$

Scoping $N \,\mathsf{sco}\, \rho$ has the same places as N and, for each potential step of N, one creates a new transition representing a synchronisation of two or more actions of N, if the potential step is not a singleton. After that, all the transitions of N are removed. Formally,

$$N \,\mathsf{sco}\, \rho \;=\; (\; N^{\mathsf{P}} \;,\; \mathcal{Z} \;)$$

$$\text{where } \mathcal{Z} = \{t \mid \exists\, U \;:\; (U, t) \in \rho_N\}.$$

As an example, the diagram of $[\![N_a \circledast (N_b \,|\!|\, N_c) \circledast N_d]\!] \,\mathsf{sco}\, \{a \mapsto g, bc \mapsto a, c \mapsto c\}$ is depicted in Figure 4(b).

As argued in [13], suitable synchronisation relations can capture a wide range of synchronisation schemes of two or more actions (e.g., $bc \mapsto a$ above), as well as action relabelling (e.g., $a \mapsto g$ and $c \mapsto c$) and action restriction (as with d).

2.4. From expressions to boxes

We can now define the semantics of box expressions by transforming them compositionally into the corresponding box nets, and then adopting the execution semantics of the latter. Formally, we define a mapping $\mathsf{box}(.)$ from expressions to boxes, in the following way:

$$\begin{array}{rcl} \mathsf{box}(\mathsf{stop}) & = & N_{\mathsf{stop}} \\ \mathsf{box}(a) & = & N_a \\ \mathsf{box}(E \,;\, E') & = & \mathsf{box}(E) \,;\, \mathsf{box}(E') \\ \mathsf{box}(E \,\Box\, E') & = & \mathsf{box}(E) \,\Box\, \mathsf{box}(E') \\ \mathsf{box}(E \,|\!|\, E') & = & \mathsf{box}(E) \,|\!|\, \mathsf{box}(E') \\ \mathsf{box}([\![E \circledast E' \circledast E'']\!]) & = & [\![\mathsf{box}(E) \circledast \mathsf{box}(E') \circledast \mathsf{box}(E'')]\!] \\ \mathsf{box}(E \,\mathsf{sco}\, \rho) & = & \mathsf{box}(E) \,\mathsf{sco}\, \rho \end{array} \qquad (3)$$

From now on, by a box we will mean a composite box constructed using (3). According to the BA theory, such boxes enjoy a number of interesting behavioural properties when we consider executions starting from their initial markings as follows (the facts listed below follow from the general results proved in [13] for more Box Algebra models):

- The number of tokens on any place for any reachable marking is bounded; more precisely, the number of tokens on any internal place is never greater than two (i.e., the internal places are 2-bounded), and an entry or exit place never holds more than one token (i.e., the entry and exit places are 1-bounded).
- Each box is clean, which means that when all the exit places contain tokens, there is no token left elsewhere in the net.
- Boxes do not allow auto-concurrency, which means that there is no transition enabled 'twice' for any reachable marking. As a consequence, steps can be represented by sets rather than by multiset of transitions.

The complexity of semantical representations of box expressions provided by the $\mathsf{box}(.)$ mapping is in the worst case exponential. To show this, let us consider a synchronised expression $F = E \,\mathsf{sco}\, \rho$. We define its size $|F|$ as $|E| + |\rho|$, where $|E|$ is the size of E taken to be the total number of occurrences of the stop's and a's within E, and $|\rho|$ to be the total number of action occurrences within $|\rho|$. Moreover, we define the size of a box N as $|N| = |N^{\mathsf{P}}| + |N^{\mathsf{T}}| + |N^{\mathsf{F}}|$. We first observe that $|\mathsf{box}(E)^{\mathsf{P}}|$, $|\mathsf{box}(E)^{\mathsf{T}}|$, and $|\mathsf{box}(E)^{\mathsf{F}}|$ all belong to $O(2^{|E|})$. Hence $|\mathsf{box}(E)| \in O(2^{|E|})$. As far as F is concerned, we observe that $|\mathsf{box}(F)^{\mathsf{P}}| = |\mathsf{box}(E)^{\mathsf{P}}| \in O(2^{|E|})$, $|\mathsf{box}(F)^{\mathsf{T}}| \in O(2^{|E|} \cdot |\rho|)$, and so $|\mathsf{box}(F)^{\mathsf{F}}|$ also belongs to $O(2^{|E|} \cdot |\rho|)$. Hence $|\mathsf{box}(F)| \in O(2^{|E|} \cdot |\rho|)$. These estimates cannot be much improved which can be shown by taking a sequence of synchronised box expressions $F_n = E_n \,\mathsf{sco}\, \rho_n$ where, for every $n \geq 1$:

$$E_n = (\underbrace{(b \parallel b) \,\square\, \ldots \,\square\, (b \parallel b)}_{n \text{ times}}) \parallel (\underbrace{a \parallel \ldots \parallel a}_{n \text{ times}}) \quad \text{and} \quad \rho_n = \{b \mapsto b\} \cup \{\underbrace{a \ldots a}_{k \text{ times}} \mapsto a \mid 1 \leq k \leq n\}.$$

We first observe that $|E_n| = 3 \cdot n$, $|\rho_n| = 2 + \frac{1}{2} \cdot n \cdot (n+3)$, and $|F_n| = 4 \cdot n + 1$. On the other hand:

$$\begin{array}{lcllllcll}
|\mathsf{box}(E_n)^{\mathsf{P}}| & = & 2^{n+1} & + & 2 \cdot n & |\mathsf{box}(E_n)^{\mathsf{P}}| & = & 2^{n+1} & + & 2 \cdot n \\
|\mathsf{box}(E_n)^{\mathsf{T}}| & = & 2 \cdot n & + & n & |\mathsf{box}(F_n)^{\mathsf{T}}| & = & 2 \cdot n & + & 2^n - 1 \\
|\mathsf{box}(E_n)^{\mathsf{F}}| & = & n \cdot 2^{n+1} & + & 2 \cdot n & |\mathsf{box}(F_n)^{\mathsf{F}}| & = & n \cdot 2^{n+1} & + & \sum_{k=1}^{n} k \cdot \binom{n}{k}.
\end{array}$$

2.5. Behavioural properties of composite boxes

Step sequences of composite boxes exhibit compositional properties [13], i.e., one can (easily) derive the semantics of a composite box from the semantics of the boxes being composed. This is demonstrated by a series of results which follow from the general properties of boxes [13].

For boxes modelling the blocking expression and a single-action expression, the semantics capture is straightforward.

Proposition 2.2. (basic boxes)
The following hold, where $a \in \mathcal{A}$:

$$\begin{array}{lcllcl} \mathsf{infsts}(N_{\mathsf{stop}}) & = & \varnothing^\omega & \mathsf{infsts}(N_a) & = & \varnothing^\omega \cup \varnothing^*\cdot\{\{a_\epsilon\}\}\cdot\varnothing^\omega \\ \mathsf{finsts}(N_{\mathsf{stop}}) & = & \varnothing & \mathsf{finsts}(N_a) & = & \varnothing^*\cdot\{\{a_\epsilon\}\}\cdot\varnothing^*. \end{array}$$

Proof:
Follows from the general results proved, e.g., in [13]. □

For choice and parallelism, the semantics of a composite box can easily be expressed in terms of the semantics of the composed boxes. We need, however, a notion of parallel composition of step sequences.

For two step sequences, θ and δ, of equal length, $\theta \parallel \delta$ is a step sequence of the same length as θ and δ, and $(\theta \parallel \delta)_{(k)} = \theta_{(k)} \cup \delta_{(k)}$, for all $k \leq |\delta| = |\theta|$. Moreover, for two sets of step sequences, Θ and Δ, $\Theta \parallel \Delta = \{\theta \parallel \delta \mid \theta \in \Theta \wedge \delta \in \Delta \wedge |\delta| = |\theta|\}$.

Proposition 2.3. (choice and parallelism)
The following hold, where *sts* = infsts or *sts* = finsts:

$$\begin{array}{rcl} \mathit{sts}(N \,\square\, K) & = & \square_L\cdot\mathit{sts}(N) \;\cup\; \square_R\cdot\mathit{sts}(N) \\ \mathit{sts}(N \parallel K) & = & \shortparallel_L\cdot\mathit{sts}(N) \;\parallel\; \shortparallel_R\cdot\mathit{sts}(N). \end{array}$$

Proof:
Follows from the general results proved, e.g., in [13]. □

Proposition 2.4. (sequence)
The following hold:

$$\begin{array}{rcl} \mathsf{infsts}(N\,;K) & = & ;_L\cdot\mathsf{finsts}(N)\cdot\,;_R\cdot\mathsf{infsts}(K) \;\cup\; ;_L\cdot\mathsf{infsts}(N) \\ \mathsf{finsts}(N\,;K) & = & ;_L\cdot\mathsf{finsts}(N)\cdot\,;_R\cdot\mathsf{finsts}(K). \end{array}$$

Proof:
Follows from the general results proved, e.g., in [13]. □

Proposition 2.5. (iteration)
The following hold:

$$\begin{array}{rcl} \mathsf{infsts}([\![N \circledast K \circledast J]\!]) & = & \circledast_L\cdot\mathsf{infsts}(N) \cup \\ & & \circledast_L\cdot\mathsf{finsts}(N)\cdot(\,\circledast_M\cdot\mathsf{finsts}(K))^*\cdot\;\circledast_M\cdot\mathsf{infsts}(K) \cup \\ & & \circledast_L\cdot\mathsf{finsts}(N)\cdot(\,\circledast_M\cdot\mathsf{finsts}(K))^*\cdot\;\circledast_R\cdot\mathsf{infsts}(J) \\ \mathsf{finsts}([\![N \circledast K \circledast J]\!]) & = & \circledast_L\cdot\mathsf{finsts}(N)\cdot(\,\circledast_M\cdot\mathsf{finsts}(K))^*\cdot\;\circledast_R\cdot\mathsf{finsts}(J). \end{array}$$

Proof:
Follows from the general results proved, e.g., in [13]. □

Finally, we consider a box $N \,\mathsf{sco}\, \rho$, where N is constructed from some non-synchronised box expression. In this case, relating step sequences of $N \,\mathsf{sco}\, \rho$ and N is more involved.

First, we define a relation $\widetilde{\rho}_N$ comprising all pairs $(U, \{t_1, \ldots, t_k\})$ $(k \geq 0)$, where $U \in N^{\mathsf{psteps}}$ and $\{t_1, \ldots, t_k\} \subseteq \mathsf{box}(N \,\mathsf{sco}\, \rho)^{\mathsf{T}}$ are such that there is a partition $U_1, \ldots, U_k$ of U satisfying $(U_j, t_j) \in \rho_N$, for each $j \leq k$. Moreover, for two equal length sequences of sets of transitions, τ and θ, we denote $(\tau, \theta) \in \widetilde{\rho}_N$ if $(\tau_{(j)}, \theta_{(j)}) \in \widetilde{\rho}_N$, for all j.

Proposition 2.6. (scoping)
sts$(N \,\mathsf{sco}\, \rho) = \{\theta \mid \exists\, \tau \in \mathit{sts}(N) \;:\; (\tau, \theta) \in \widetilde{\rho}_N\}$, where *sts* $= \mathsf{infsts}$ or *sts* $= \mathsf{finsts}$.

Proof:
Follows from the general results proved, e.g., in [13]. □

2.6. Streamlined box expressions

The translation from BA to ITL, that will be described in Section 4, is particularly simple for the class of streamlined synchronised expressions.

We first observe that each transition a_W in the box associated with a non-synchronized box expression is such that W is a singleton. Thus, a_W is represented as a_π for a syntax path π.

We call a box expression $E \,\mathsf{sco}\, \rho$ *streamlined* if for each transition $a_\pi \in \mathsf{box}(E)^{\mathsf{T}}$ there is exactly one transition $b_W \in \mathsf{box}(E \,\mathsf{sco}\, \rho)^{\mathsf{T}}$ such that $\pi \in W$. That is, any pre-synchronisation action in a streamlined box expression contributes to *exactly one* action after applying the synchronisation.[3]

We will now demonstrate that each synchronised box expression $F = E \,\mathsf{sco}\, \rho$ can be transformed into a semantically equivalent[4] streamlined expression $\mathsf{stl}(F) = F' = E' \,\mathsf{sco}\, \rho'$. First, for every $a_\pi \in \mathsf{box}(E)^{\mathsf{T}}$, let

$$\mathsf{trans}(a_\pi) = \{b_W \in \mathsf{box}(F)^{\mathsf{T}} \mid \pi \in W\} = \{t \mid \exists (U, t) \in \rho_{\mathsf{box}(E)} \;:\; a_\pi \in U\}.$$

In other words, $\mathsf{trans}(a_\pi)$ comprises all post-synchronisation transitions in which a_π has been involved. For example, if we take $[\![a \circledast (b \parallel c) \circledast d]\!] \,\mathsf{sco}\, \{a \mapsto g, bc \mapsto a, c \mapsto c\}$ with the corresponding box depicted in Figure 4(b), we have:

$$\begin{array}{lcllcl} \mathsf{trans}(a_{\circledast_L}) & = & \{g_{\circledast_L}\} & \mathsf{trans}(b_{\circledast_{M \parallel L}}) & = & \{a_{\circledast_{M \parallel L}}, {\circledast_{M \parallel R}}\} \\ \mathsf{trans}(d_{\circledast_R}) & = & \varnothing & \mathsf{trans}(c_{\circledast_{M \parallel R}}) & = & \{a_{\circledast_{M \parallel L}}, {\circledast_{M \parallel R}}, c_{\circledast_{M \parallel R}}\} \end{array}$$

Then, a suitable E' is obtained by replacing each occurrence of an action $a \in \mathcal{A}$ in E corresponding to transition a_π in $\mathsf{box}(E)$[5] by:

- stop if we have $\mathsf{trans}(a_\pi) = \varnothing$,
- b_W if we have $\mathsf{trans}(a_\pi) = \{b_W\}$, and
- $b^1_{W_1} \,\square\, (\ldots \,\square\, (b^{m-1}_{W_{m-1}} \,\square\, b^m_{W_m}) \ldots)$ if we have $\mathsf{trans}(a_\pi) = \{b^1_{W_1}, \ldots, b^m_{W_m}\}$ and $m \geq 2$[6].

[3] As a result, each pre-synchronisation action can be represented by a single variable in the corresponding ITL formula.
[4] In the sense of generating isomorphic box net with 'isomorphic' step sequences, see Proposition 2.7.
[5] Such an occurrence of a is identified by the path in the syntax tree of the non-synchronised expression E which corresponds to π.
[6] We assume a fixed ordering on the transitions of $\mathsf{box}(F)$ so that the enumeration of $\mathsf{trans}(a_\pi)$ is unique.

Furthermore, we modify the scoping part reflecting changes affecting the set of actions:

$$\rho' = \{\underbrace{b_W \dots b_W}_{|W| \text{ times}} \mapsto b_W \mid b_W \in \mathsf{box}(F)^{\mathsf{T}}\}$$

The $b_W \dots b_W \mapsto b_W$ above reflects the fact that b_W is constructed as a synchronisation of $|W|$ actions in $\mathsf{box}(E)$, each such action being now replaced by a copy of b_W in E'.

Continuing the last example, we obtain $F' = E' \,\mathsf{sco}\, \rho'$, with

$$E' = (\llbracket \gamma \circledast (\alpha \parallel (\alpha \,\Box\, \zeta)) \circledast \mathsf{stop} \rrbracket) \quad \text{and} \quad \rho' = \{\gamma \mapsto \gamma, \alpha\alpha \mapsto \alpha, \zeta \mapsto \zeta\},$$

where $\gamma = g_{\circledast_L}$, $\alpha = a_{\circledast_{M^{\shortmid\shortmid}L}, \circledast_{M^{\shortmid\shortmid}R}}$, and $\zeta = c_{\circledast_{M^{\shortmid\shortmid}R}}$.

Each transition of $\mathsf{box}(F')$ is of the form $(b_W)_Y$, where $b_W \in \mathsf{box}(E \,\mathsf{sco}\, \rho)^{\mathsf{T}}$,[7] and we then define a bijection $\lambda : \mathsf{box}(F')^{\mathsf{T}} \to \mathsf{box}(F)^{\mathsf{T}}$ by setting $\lambda((b_W)_Y) = b_W$. Such a λ can be applied in the standard way to the subsets of $\mathsf{box}(F')^{\mathsf{T}}$ and their sequences.

Proposition 2.7. (streamlined expression)
The nets $\mathsf{box}(F)$ and $\mathsf{box}(\mathsf{stl}(F))$ are isomorphic after replacing each transition label b_W in $\mathsf{box}(\mathsf{stl}(F))$ by b. Moreover, $\mathit{sts}(\mathsf{box}(F)) = \lambda(\mathit{sts}(\mathsf{box}(\mathsf{stl}(F))))$, where $\mathit{sts} = \mathsf{infsts}$ or $\mathit{sts} = \mathsf{finsts}$.

Proof:
We observe that the sets of places in $\mathsf{box}(F)$ and $\mathsf{box}(\mathsf{stl}(F))$ are in a one-to-one correspondence since the subexpressions of the form stop or b_W or $b^1_{W_1} \,\Box\, (\dots \,\Box\, (b^{m-1}_{W_{m-1}} \,\Box\, b^m_{W_m}) \dots)$ produce boxes with exactly one entry place, one exit place, and no internal places.

The first part of the result follows from the fact that the result of synchronising $|W|$ transitions in $\mathsf{box}(E)$ to yield a transition b_W is the same w.r.t. connections with the (corresponding) places as the result of synchronising $|W|$ transitions labelled b_W in $\mathsf{box}(E')$ yielding a transition $(b_W)_Y$.

The second part follows from the first part and the fact that λ is a part of isomorphism between $\mathsf{box}(F)$ and $\mathsf{box}(\mathsf{stl}(F))$. □

Streamlined expressions will prove their usefulness in the translation of box expressions to behaviourally equivalent ITL formulas. Intuitively, they allow one to separate the roles that a single variable can play in different synchronisation contexts.

In terms of complexity, the transformation of $F = E \,\mathsf{sco}\, \rho$ into a streamlined $\mathsf{stl}(F) = E' \,\mathsf{sco}\, \rho'$ will usually result in a larger expression. This increase is a result of inserting sub-expressions of the form $b^1_{W_1} \,\Box\, (\dots \,\Box\, (b^{m-1}_{W_{m-1}} \,\Box\, b^m_{W_m}) \dots)$ and modifying the scoping relation. It is straightforward to check that $|E'| \in O(|E| + |\mathsf{box}(E)^{\mathsf{F}}|)$ and $|\rho'| = |\mathsf{box}(E)^{\mathsf{F}}|$. Hence $|\mathsf{stl}(F)| \in O(|E| + |\mathsf{box}(E)^{\mathsf{F}}|)$, and so $|\mathsf{stl}(F)| \in O(|E| + |\mathsf{box}(E)|)$. Since it can be argued that, for the purpose of behavioural analyses within the domain of Petri nets, the problem size is $|F| + |\mathsf{box}(F)|$, the transformation to streamlined expressions is efficient.
It is also important to stress that it is not necessary to derive $\mathsf{box}(F)$ in order to derive $\mathsf{stl}(F)$.

[7] In general, $W \neq Y$ since the syntax tree of E is not isomorphic to the syntax tree of E' due to the introduction of sub-expressions of the form $b^1_{W_1} \,\Box\, (\dots \,\Box\, (b^{m-1}_{W_{m-1}} \,\Box\, b^m_{W_m}) \dots)$.

3. Interval Temporal Logic

We now provide the syntax and semantics of a small fragment of ITL, including only those constructs (basic and derived) which are used in the translation of box expressions. In particular, we assume that Var is a countable set of Boolean variables, all such variables being the *transitions* of boxes created using the box mapping.

The formulas of the fragment of the ITL logic we need are defined by:

$$\varphi ::= \mathsf{true} \mid \mathsf{flip}(v) \mid \mathsf{skipstbl}(v) \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi ; \varphi \mid \varphi^* \mid \mathsf{inf}$$

where $v \in Var$. Intuitively, $\mathsf{flip}(v)$ inverts the value of a Boolean variable v over a unit interval, $\mathsf{skipstbl}(v)$ keeps unchanged the value of v over a unit interval, "$;$" is a sequential composition operator (called chop), "*" is an iterative version of chop (called chop-star), and inf indicates an infinite interval. The set of variables occurring in a formula φ is denoted by $\mathsf{var}(\varphi)$. and its size $|\varphi|$ is the total number of occurrences of the v's, stop's, and inf's within φ.

Remark 3.1. The logic syntax introduced above has been tailored to smooth the translation from box expression to logical formulas. However, all the non-standard constructs used (i.e., true, flip, skipstbl and inf) can be expressed in the standard ITL logic, in the following way:

$$\begin{array}{rclrcl} \mathsf{true} & = & v \vee \neg v & \mathsf{flip}(v) & = & (\bigcirc\neg \bigcirc \mathsf{true}) \wedge (v = \bigcirc\neg v) \\ \mathsf{inf} & = & \mathsf{true}\,;\neg\mathsf{true} & \mathsf{skipstbl}(v) & = & (\bigcirc\neg \bigcirc \mathsf{true}) \wedge (v = \bigcirc v) \end{array}$$

where $\bigcirc$ is the temporal operator 'next'. Hence it is possible to formulate and analyse behavioural properties of any translated formula using proof techniques and tools of ITL (e.g., [11] gives a complete axiomatisation of both finite and infinite time). $\diamondsuit$

A *state* s is a mapping which assigns values to the Boolean variables Var, and an *interval* σ is a non-empty sequence of states. The meaning of formulas is given by the satisfaction relation $\models$ involving intervals and formulas, defined as follows (below s, s' are states, and $\sigma, \sigma', \sigma''$ are intervals):

- $\sigma \models \mathsf{true}$,
- $\sigma \models \mathsf{flip}(v)$ if $\sigma = s{\cdot}s'$ and $s(v) \neq s'(v)$,
- $\sigma \models \mathsf{skipstbl}(v)$ if $\sigma = s{\cdot}s'$ and $s(v) = s'(v)$,
- $\sigma \models \varphi \vee \varphi'$ if $\sigma \models \varphi$ or $\sigma \models \varphi'$,
- $\sigma \models \varphi \wedge \varphi'$ if $\sigma \models \varphi$ and $\sigma \models \varphi'$,
- $\sigma \models \varphi ; \varphi'$ if σ is infinite and $\sigma \models \varphi$, or σ can be decomposed as $\sigma = \sigma'{\cdot}s{\cdot}\sigma''$ so that $\sigma'{\cdot}s \models \varphi$ and $s{\cdot}\sigma'' \models \varphi'$,
- $\sigma \models \varphi^*$ if $|\sigma| = 1$, or σ can be decomposed as $\sigma = \sigma'{\cdot}s{\cdot}\sigma''$ so that $\sigma'{\cdot}s \models \varphi$ and $s{\cdot}\sigma'' \models \varphi^*$, and

- $\sigma \models \mathsf{inf}$ if σ is infinite.

To capture the relationship between the semantics of a box expression and a corresponding formula, with each ITL formula φ and interval σ satisfying $\sigma \models \varphi$, we associate a sequence of sets $\mu_\sigma = \Gamma_1\Gamma_2\ldots$, where, for each $j < |\sigma|$, Γ_j is given by:

$$\Gamma_j = \mathsf{var}(\varphi) \cap \{v \in \mathsf{var}(\varphi) \mid \sigma_(j)(v) \neq \sigma_{(j+1)}(v)\}.$$

Since each Γ_j records all the variables which flipped their values at the point of entering the state $\sigma_{(j+1)}$, the sequence μ_σ provides a direct interpretation of σ in terms of sequences of steps of transitions of box nets. Then, for any ITL formula φ, we define:

$$\mathsf{infsts}(\varphi) = \{\mu_\sigma \mid \sigma \models \varphi \wedge |\sigma| = \omega\} \quad \text{and} \quad \mathsf{finsts}(\varphi) = \{\mu_\sigma \mid \sigma \models \varphi \wedge |\sigma| < \omega\}.$$

Below we present a number of semantrical properties of ITL formulas considered in this paper.

Proposition 3.2. Let φ and φ' be two formulas with disjoint sets of variables, i.e., $\mathsf{var}(\varphi) \cap \mathsf{var}(\varphi') = \varnothing$. Then, the following hold, where $\psi = \mathsf{skipstbl}(\mathsf{var}(\varphi))^*$, $\psi' = \mathsf{skipstbl}(\mathsf{var}(\varphi'))^*$, and $sts = \mathsf{infsts}$ or $sts = \mathsf{finsts}$:

$$\begin{array}{rcl}
sts((\varphi \wedge \psi') \vee (\varphi' \wedge \psi)) & = & sts(\varphi) \cup sts(\varphi') \\
sts(\varphi \wedge \varphi') & = & sts(\varphi) \parallel sts(\varphi') \\
\mathsf{infsts}((\varphi \wedge \psi')\,;(\varphi' \wedge \psi)) & = & \mathsf{infsts}(\varphi) \cup \mathsf{finsts}(\varphi)\cdot\mathsf{infsts}(\varphi') \\
\mathsf{finsts}((\varphi \wedge \psi')\,;(\varphi' \wedge \psi)) & = & \mathsf{finsts}(\varphi)\cdot\mathsf{finsts}(\varphi').
\end{array}$$

Proof:
Follows directly from the basic properties of logic operators. □

Proposition 3.3. Let φ_i, for $i = 1, 2, 3$, be formulas with mutually disjoint sets of variables. Then, the following hold, where $\psi_{i,j} = \mathsf{skipstbl}(\mathsf{var}(\varphi_i) \cup \mathsf{var}(\varphi_j))^*$, for $i, j \in \{1, 2, 3\}$:

$$\begin{array}{rcl}
\mathsf{finsts}((\varphi_1 \wedge \psi_{2,3})\,;((\varphi_2 \wedge \psi_{1,3})^*\,;(\varphi_3 \wedge \psi_{1,2}))) & = & \mathsf{finsts}(\varphi_1)\cdot\mathsf{finsts}(\varphi_2)^*\cdot\mathsf{finsts}(\varphi_3) \\
\mathsf{infsts}((\varphi_1 \wedge \psi_{2,3})\,;((\varphi_2 \wedge \psi_{1,3})^*\,;(\varphi_3 \wedge \psi_{1,2}))) & = & \mathsf{infsts}(\varphi_1) \cup \\
 & & \mathsf{finsts}(\varphi_1)\cdot\mathsf{finsts}(\varphi_2)^*\cdot(\mathsf{infsts}(\varphi_2) \cup \mathsf{infsts}(\varphi_3)).
\end{array}$$

Proof:
We obtain the following, after noting that $\mathsf{var}(\varphi_2^*) = \mathsf{var}(\varphi_2)$ and twice using Proposition 3.2:

$$\begin{array}{l}
\mathsf{finsts}((\varphi_1 \wedge \psi_{2,3})\,;((\varphi_2 \wedge \psi_{1,3})^*\,;(\varphi_3 \wedge \psi_{1,2})) \\
\quad = \mathsf{finsts}((\varphi_1 \wedge \psi_{2,3})\,;(((\varphi_2 \wedge \psi_{3,3})^*\,;(\varphi_3 \wedge \psi_{2,2})) \wedge \psi_{1,1}) \\
\quad = \mathsf{finsts}(\varphi_1)\cdot\mathsf{finsts}((\varphi_2 \wedge \psi_{3,3})^*\,;(\varphi_3 \wedge \psi_{2,2})) \\
\quad = \mathsf{finsts}(\varphi_1)\cdot\mathsf{finsts}((\varphi_2^* \wedge \psi_{3,3})\,;(\varphi_3 \wedge \mathsf{skipstbl}(\mathsf{var}(\varphi_2^*))^*)) \\
\quad = \mathsf{finsts}(\varphi_1)\cdot\mathsf{finsts}(\varphi_2^*)\cdot\mathsf{finsts}(\varphi_3) \\
\quad = \mathsf{finsts}(\varphi_1)\cdot\mathsf{finsts}(\varphi_2)^*\cdot\mathsf{finsts}(\varphi_3).
\end{array}$$

The second part of the proof is similar. □

Proposition 3.4. Let φ' be a formula obtained from an ITL formula φ by a consistent renaming of Boolean variables given by a bijection λ. Then $\mathit{sts}(\varphi') = \lambda(\mathit{sts}(\varphi))$, for $\mathit{sts} = \mathsf{infsts}$ or $\mathit{sts} = \mathsf{finsts}$.

Proof:
Follows from the insensitivity of $\models$ to the identities of logic variables. □

For a formula φ and $\pi \in \Pi$, we will denote by $\pi{\cdot}\varphi$ the formula obtained from φ by replacing each variable $a_{\pi'}$ with $a_{\pi\cdot\pi'}$.

Proposition 3.5. If $\pi \in \Pi$ then $\mathit{sts}(\pi{\cdot}\varphi) = \pi{\cdot}\mathit{sts}(\varphi)$, where $\mathit{sts} = \mathsf{infsts}$ or $\mathit{sts} = \mathsf{finsts}$.

Proof:
This follows from Proposition 3.4 and the fact that the transformation given by $\pi{\cdot}\varphi$ is a consistent renaming of variables. □

3.1. Derived formulas

and we will use the following derived formulas, for finite sets of variables $V, V' \subseteq \mathit{Var}$:

$$\begin{array}{rcl}
\mathsf{skipstbl}(V) & = & \begin{cases} \bigwedge_{v\in V} \mathsf{skipstbl}(v) & \text{if } V \neq \varnothing \\ \mathsf{true} & \text{otherwise} \end{cases} \\
\mathsf{infstbl}(V) & = & \mathsf{inf} \wedge \mathsf{skipstbl}(V)^* \\
\mathsf{flip}(V) & = & \begin{cases} \bigvee_{v\in V} \mathsf{flip}(v) \wedge \mathsf{skipstbl}(V \setminus \{v\}) & \text{if } V \neq \varnothing \\ \mathsf{inf} & \text{otherwise} \end{cases} \\
\mathsf{fs}(V \mid V') & = & \mathsf{skipstbl}(V \cup V')^* \,;\, (\mathsf{flip}(V) \wedge \mathsf{skipstbl}(V')) \,;\, \mathsf{skipstbl}(V \cup V')^*.
\end{array} \tag{4}$$

Intuitively, $\mathsf{skipstbl}(V)$ and $\mathsf{infstbl}(V)$ keep unchanged the values of the variables in V respectively over a unit and infinite interval, $\mathsf{flip}(V)$ inverts the value of exactly one of the Boolean variables in V over a unit interval, and $\mathsf{fs}(V \mid V')$ keeps unchanged the values of the variables in V and V' over an interval except that it inverts the value of exactly one variable in V over a unit sub-interval. We can drop the set parenthesis when writing down the above formulas.

Using the derived syntax, we call an ITL logical formula an *infstbl/fs*-formula if it constructed using $\mathsf{infstbl}(V)$'s, $\mathsf{fs}(V \mid V')$'s and the four logical operators of the syntax. The translation we will introduce in the next section will translate box expressions into *infstbl/fs*-formulas.

It is easily checked that the following hold for any finite sets of variables $V, V', V'' \subseteq \mathit{Var}$ satisfying $V \cap V'' = \varnothing$, where $\equiv$ denotes the standard equivalence of formulas w.r.t. the satisfaction relation $\models$:

$$\begin{array}{rclcrcl}
\mathsf{fs}(\varnothing \mid V) & \equiv & \mathsf{infstbl}(V) & \qquad & \mathsf{infstbl}(V) \wedge \mathsf{skipstbl}(V')^* & \equiv & \mathsf{infstbl}(V \cup V') \\
\mathsf{fs}(V \mid V) & \equiv & \mathsf{infstbl}(V) & & \mathsf{fs}(V \mid V') \wedge \mathsf{skipstbl}(V'')^* & \equiv & \mathsf{fs}(V \mid V' \cup V'').
\end{array} \tag{5}$$

Moreover, $\mathsf{skipstbl}(V)$ distributes over the logical operators as we have the following, for all formulas φ, φ' and finite sets of variables V:

$$\begin{array}{lcl} (\varphi \wedge \varphi') \wedge \mathsf{skipstbl}(V) & \equiv & (\varphi \,;\, \varphi') \wedge \mathsf{skipstbl}(V) \\ (\varphi \wedge \varphi') \vee \mathsf{skipstbl}(V) & \equiv & (\varphi \wedge \mathsf{skipstbl}(V)) \vee (\varphi' \wedge \mathsf{skipstbl}(V)) \\ (\varphi \wedge \varphi') \,;\, \mathsf{skipstbl}(V) & \equiv & (\varphi \wedge \mathsf{skipstbl}(V)) \,;\, (\varphi' \wedge \mathsf{skipstbl}(V)). \end{array} \tag{6}$$

For every *infstbl/fs*-formula φ and a finite set of variables V, we will denote by $V \blacktriangleright \varphi$ the formula obtained by replacing each sub-formula $\mathsf{infstbl}(V')$ by $\mathsf{infstbl}(V \cup V')$, and each sub-formula $\mathsf{fs}(V' \,|\, V'')$ by $\mathsf{fs}(V' \,|\, V \cup V'')$. By (5) and (6), we immediately obtain that:

$$V \blacktriangleright \varphi \equiv \varphi \wedge \mathsf{skipstbl}(V)^*. \tag{7}$$

4. From box expressions to logical formulas

To make the presentation more accessible, we will first show how to translate non-synchronised box expressions. After that, we will extend the translation to the streamlined synchronised expressions, and, finally, we will deal with the case of general synchronised expressions.

4.1. Translating non-synchronised expressions

The translation for non-synchronised expressions yields *infstbl/fs*-formulas, and is defined compositionally in the following way:

$$\begin{array}{rcl} \mathsf{itl}(\mathsf{stop}) & = & \mathsf{infstbl}(\varnothing) \\ \mathsf{itl}(a) & = & \mathsf{fs}(a_\epsilon \,|\, \varnothing) \\ \mathsf{itl}(E \,;\, F) & = & ((\,;_R \cdot \mathsf{var}(\mathsf{itl}(F))) \blacktriangleright (\,;_L \cdot \mathsf{itl}(E))) \;;\; ((\,;_L \cdot \mathsf{var}(\mathsf{itl}(E))) \blacktriangleright (\,;_R \cdot \mathsf{itl}(F))) \\ \mathsf{itl}(E \,\Box\, F) & = & ((\,\Box_R \cdot \mathsf{var}(\mathsf{itl}(F))) \blacktriangleright (\,\Box_L \cdot \mathsf{itl}(E))) \;\vee\; ((\,\Box_L \cdot \mathsf{var}(\mathsf{itl}(E))) \vee (\,\Box_R \cdot \mathsf{itl}(F))) \\ \mathsf{itl}(E \parallel F) & = & \shuffle_L \cdot \mathsf{itl}(E) \;\wedge\; \shuffle_R \cdot \mathsf{itl}(F) \\ \mathsf{itl}([\![E \circledast F \circledast G]\!]) & = & (((\circledast_M \cdot \mathsf{var}(\mathsf{itl}(F))) \cup (\circledast_R \cdot \mathsf{var}(\mathsf{itl}(G)))) \blacktriangleright (\circledast_L \cdot \mathsf{itl}(E))) \;; \\ & & (((\circledast_L \cdot \mathsf{var}(\mathsf{itl}(E))) \cup (\circledast_R \cdot \mathsf{var}(\mathsf{itl}(G)))) \blacktriangleright (\circledast_M \cdot \mathsf{itl}(F)))^* \;; \\ & & (((\circledast_L \cdot \mathsf{var}(\mathsf{itl}(E))) \cup (\circledast_M \cdot \mathsf{var}(\mathsf{itl}(F)))) \blacktriangleright (\circledast_R \cdot \mathsf{itl}(G))). \end{array}$$

As an example, for the expressions generating the boxes in Figure 3, we obtain:

$$\begin{array}{rcl} \mathsf{itl}(a \,;\, b) & = & \mathsf{fs}(a_{\,;_L} \,|\, b_{\,;_R}) \,;\, \mathsf{fs}(b_{\,;_R} \,|\, a_{\,;_L}) \\ \mathsf{itl}(c \,\Box\, (a \,;\, b)) & = & \big(\mathsf{fs}(c_{\,\Box_L} \,|\, a_{\,\Box_R ;_L}, b_{\,\Box_R ;_R})\big) \vee \big(\mathsf{fs}(a_{\,\Box_R ;_L} \,|\, b_{\,\Box_R ;_R}, c_{\,\Box_L}) \,;\, \mathsf{fs}(b_{\,\Box_R ;_R} \,|\, a_{\,\Box_R ;_L}, c_{\,\Box_L})\big). \end{array}$$

As far as the complexity of the translation is concerned, $|\mathsf{itl}(E)| \in O(|E|^2)$. This compares very favourably with the size of the original semantical representation of E since $|\mathsf{box}(E)| \in O(2^{|E|})$.

Crucially, the step semantics of a non-synchronised box expression and the corresponding ITL formula coincide.

Theorem 4.1. (non-synchronised expression)
If H is a non-synchronised box expression then $sts(\mathsf{itl}(H)) = sts(\mathsf{box}(H))$, where $sts = \mathsf{infsts}$ or $sts = \mathsf{finsts}$.

Proof:
Let $\psi = \mathsf{itl}(H)$ and $N = \mathsf{box}(H)$. The proof proceeds by induction on the structure of H.
Case 1: $H = \mathsf{stop}$. Then $\psi = \mathsf{inf} \equiv \mathsf{infstbl}(\varnothing)$ and $N = N_{\mathsf{stop}}$. Hence we have:

$$\mathsf{infsts}(\psi) = \varnothing^{\omega} = \mathsf{infsts}(N) \quad \text{and} \quad \mathsf{finsts}(\psi) = \varnothing = \mathsf{finsts}(N).$$

Case 2: $H = a$. Then, $\psi = \mathsf{fs}(a_\epsilon \,|\, \varnothing) = \mathsf{skipstbl}(a_\epsilon)^* \,;\, \mathsf{flip}(a_\epsilon) \,;\, \mathsf{skipstbl}(a_\epsilon)^*$ and $N = N_a$. Hence we have:

$$\mathsf{infsts}(\psi) = \varnothing^{\omega} \;\cup\; \varnothing^* \cdot \{\{a_\epsilon\}\} \cdot \varnothing^{\omega} = \mathsf{infsts}(N)$$
$$\mathsf{finsts}(\psi) = \varnothing^* \cdot \{\{a_\epsilon\}\} \cdot \varnothing^* = \mathsf{finsts}(N).$$

Case 3: $H = E \,\Box\, F$. Then, by the equivalence (7), we have:

$$\psi \;\equiv\; \Box_L \cdot \mathsf{itl}(E) \wedge \mathsf{skipstbl}(\,\Box_R \cdot \mathsf{var}(\mathsf{itl}(F)))^* \;\vee\; \Box_R \cdot \mathsf{itl}(F) \wedge \mathsf{skipstbl}(\,\Box_L \cdot \mathsf{var}(\mathsf{itl}(E)))^*.$$

Hence, by $\mathsf{var}(\psi) = \Box_L \cdot \mathsf{var}(\mathsf{itl}(E)) \cup \Box_R \cdot \mathsf{var}(\mathsf{itl}(F))$ as well as (applied in this order) Proposition 3.2, Proposition 3.5, the induction hypothesis, and Proposition 2.3, we obtain:

$$\begin{array}{rcl} sts(\psi) & = & sts(\,\Box_L \cdot \mathsf{itl}(E)) \;\cup\; sts(\,\Box_R \cdot \mathsf{itl}(F)) \\ & = & \Box_L \cdot sts(\mathsf{itl}(E)) \;\cup\; \Box_R \cdot sts(\mathsf{itl}(F)) \\ & = & \Box_L \cdot sts(\mathsf{box}(E)) \;\cup\; \Box_R \cdot sts(\mathsf{box}(F)) \;=\; sts(H). \end{array}$$

Case 4: $H = E \parallel F$. Then, by Proposition 3.2, Proposition 3.5, the induction hypothesis, and Proposition 2.3, we have:

$$\begin{array}{rcl} sts(\psi) & = & sts(\shortparallel_L \cdot \mathsf{itl}(E)) \;\parallel\; sts(\shortparallel_L \cdot \mathsf{itl}(F)) \\ & = & \shortparallel_L \cdot sts(\mathsf{itl}(E)) \;\parallel\; \shortparallel_L \cdot sts(\mathsf{itl}(F)) \\ & = & \shortparallel_L \cdot sts(\mathsf{box}(E)) \;\parallel\; \shortparallel_L \cdot sts(\mathsf{box}(F)) \;=\; sts(H). \end{array}$$

Case 5: $H = E \,;\, F$. Then, by the equivalence (7), we have:

$$\psi \;=\; ;_L \cdot \mathsf{itl}(E) \wedge \mathsf{skipstbl}(\,;_R \cdot \mathsf{var}(\mathsf{itl}(F)))^* \;\;;\;\; ;_R \cdot \mathsf{itl}(F) \wedge \mathsf{skipstbl}(\,;_L \cdot \mathsf{var}(\mathsf{itl}(E)))^*.$$

Hence we obtain, by $\mathsf{var}(\psi) = ;_L \cdot \mathsf{var}(\mathsf{itl}(E)) \cup ;_R \cdot \mathsf{var}(\mathsf{itl}(F))$ as well as (applied in this order) Proposition 3.2, Proposition 3.5, the induction hypothesis, and Proposition 2.4:

$$\begin{array}{rcl} \mathsf{finsts}(\psi) & = & \mathsf{finsts}(\,;_L \cdot \mathsf{itl}(E)) \;\cdot\; \mathsf{finsts}(\,;_R \cdot \mathsf{itl}(F)) \\ & = & ;_L \cdot \mathsf{finsts}(\mathsf{itl}(E)) \;\cdot\; ;_R \cdot \mathsf{finsts}(\mathsf{itl}(F)) \\ & = & ;_L \cdot \mathsf{finsts}(\mathsf{box}(E)) \cdot\; ;_R \cdot \mathsf{finsts}(\mathsf{box}(F)) \;=\; \mathsf{finsts}(H). \end{array}$$

The second part of the proof for sequence is similar.

Case 6: $H = [\![E \circledast F \circledast G]\!]$. Then, by the equivalence (7), we have:

$$\begin{array}{rcl} \psi & = & \circledast_L{\cdot}\mathsf{itl}(E) \ \wedge \mathsf{skipstbl}(\ \circledast_M{\cdot}\mathsf{var}(F) \cup \circledast_R{\cdot}\mathsf{var}(G)\ )^* \quad ; \\ & & (\ \circledast_M{\cdot}\mathsf{itl}(F) \wedge \mathsf{skipstbl}(\ \circledast_L{\cdot}\mathsf{var}(E)\ \cup \circledast_R{\cdot}\mathsf{var}(G)\ )^*)^* \ ; \\ & & \circledast_R{\cdot}\mathsf{itl}(G)\ \wedge \mathsf{skipstbl}(\ \circledast_L{\cdot}\mathsf{var}(E)\ \cup \circledast_M{\cdot}\mathsf{var}(F)\ )^* \end{array}$$

Hence we obtain, by $\mathsf{var}(\psi) = \circledast_L{\cdot}\mathsf{var}(\mathsf{itl}(E)) \cup \circledast_M{\cdot}\mathsf{var}(\mathsf{itl}(F)) \cup \circledast_R{\cdot}\mathsf{var}(\mathsf{itl}(G))$ as well as (applied in this order) Proposition 3.3, Proposition 3.5, the induction hypothesis, and Proposition 2.5:

$$\begin{array}{rcl} \mathsf{finsts}(\psi) & = & \mathsf{finsts}(\ \circledast_L{\cdot}\mathsf{itl}(E))\ \cdot \mathsf{finsts}(\ \circledast_R{\cdot}\mathsf{itl}(F))^* \ \cdot \mathsf{finsts}(\ \circledast_M{\cdot}\mathsf{itl}(G)) \\ & = & \circledast_L{\cdot}\mathsf{finsts}(\mathsf{itl}(E))\ \cdot (\ \circledast_R{\cdot}\mathsf{finsts}(\mathsf{itl}(F)))^* \ \cdot \circledast_M{\cdot}\mathsf{finsts}(\mathsf{itl}(G)) \\ & = & \circledast_L{\cdot}\mathsf{finsts}(\mathsf{box}(E)) \cdot (\ \circledast_R{\cdot}\mathsf{finsts}(\mathsf{box}(F)))^* \cdot \circledast_M{\cdot}\mathsf{finsts}(\mathsf{box}(G)) \ = \ \mathsf{finsts}(H). \end{array}$$

The second part of the proof for iteration is similar. □

4.2. Translating streamlined expressions

The result captured by Theorem 4.1 is strong as it means that the behavioural properties of non-synchronised box expressions related to the sequencing of executed actions can be re-interpreted as properties of the translated formulas, assuming that an execution of a transition is 'simulated' by a flipping of the corresponding Boolean variable. Extending such a result to streamlined expressions highlights the way in which the box expression synchronisation mechanism (through merging transitions) and the ITL synchronisation mechanism (through flipping variables in different parts of a formula) can be made to match each other.

Let $F = E\,\mathsf{sco}\,\rho$ be a streamlined box expression. Then, $\mathsf{itl}(F)$ is obtained from $\mathsf{itl}(E)$ by replacing each occurrence of each variable v by the unique variable in $\mathsf{trans}(v)$[8].
It is also important to stress that it is not necessary to derive $\mathsf{box}(F)$ in order to derive the $\mathsf{trans}(v)$'s.

Theorem 4.2. (streamlined expression)
Let $F = E\,\mathsf{sco}\,\rho$ be a streamlined box expression. Then, $sts(\mathsf{itl}(F)) = sts(\mathsf{box}(F))$, for $sts = \mathsf{infsts}$ or $sts = \mathsf{finsts}$.

Proof:
By the definition of the itl mapping, flipping the value of any variable v in $\mathsf{itl}(E)$ is due to the (unique within $\mathsf{itl}(E)$) sub-formula $\mathsf{flip}(v)$ as otherwise v keeps the same value due to the presence of the $\mathsf{skipstbl}(v)^*$ sub-formulas.

Suppose now that a_W is a variable in $\mathsf{itl}(F)$ and that $a^1_{\pi_1}, \ldots, a^k_{\pi_k}$ are the variables in $\mathsf{itl}(E)$ which in $\mathsf{itl}(F)$ are replaced by a_W. Then, by definition, $\pi_i | \pi_j$ for all $i \neq j$. Hence, there is a sub-formula $\varphi \wedge \varphi'$ of $\mathsf{itl}(E)$ such that $\mathsf{var}(\varphi) \cap \mathsf{var}(\varphi') = \varnothing$ and, without loss of generality, $a^i_{\pi_i} \in \mathsf{var}(\varphi)$ and $a^j_{\pi_j} \in \mathsf{var}(\varphi')$. As this observation holds for all distinct i and j, it follows that flipping of a_W in $\mathsf{itl}(F)$ must be 'agreed upon' by all the sub-formulas $\mathsf{flip}(a_W)$, each resulting from a replacement of some $a^i_{\pi_j}$ by a_W. The result then follows from $sts(\mathsf{itl}(E)) = sts(\mathsf{box}(E))$ (see Theorem 4.1) and Proposition 2.6. □

[8]Recall $\mathsf{trans}(t)$ was introduced in Section 2.6.

In terms of complexity, the size of $\mathsf{itl}(F)$ is of the same order as that $\mathsf{itl}(E)$, and so $|\mathsf{itl}(F)| \in O(|F|^2)$ which, again, compares very favourably with $|\mathsf{box}(F)|$ as $|\mathsf{box}(F)^{\mathsf{T}}| \in O(2^{|E|})$. Moreover, the number variables in $\mathsf{itl}(F)$ is the same as the number of transitions in $\mathsf{box}(F)$.

4.3. Translating general expressions

Suppose now that $F = E\,\mathsf{sco}\,\rho$ is an arbitrary synchronised expression. Given Proposition 2.7, we could now simply take the streamlined expression $\mathsf{stl}(F)$ defined in Section 2.6 and, after consistently renaming variables according to the bijection λ defined in Section 2.6, derive $\mathsf{itl}(\mathsf{stl}(F))$ and obtain a generalised version of Theorem 4.2.

Applying our previous complexity estimates, we would get $|\mathsf{itl}(\mathsf{stl}(F))| \in O((|E|+|\mathsf{box}(E)|^{\mathsf{F}})^2)$. Therefore, $\mathsf{itl}(\mathsf{stl}(F))$ would be of similar size as $\mathsf{box}(E)$. This, however, changes radically if we assumed that, e.g., only binary synchronisations are allowed and a given pair of synchronised actions always yields the same action label,[9] and so $|\mathsf{box}(E)|^{\mathsf{F}} \in O(|E|^2)$. Then we would have $|\mathsf{itl}(\mathsf{stl}(F))| \in O(|E|^4)$ whereas $|\mathsf{box}(E)|$ would belong to $O(2^{|E|})$.[10] Moreover, as we already argued, it is not necessary to construct $\mathsf{box}(F)$ in order to derive $\mathsf{itl}(\mathsf{stl}(F))$.

As an alternative, we may proceed without pre-processing and conservatively extend the translation defined for non-synchronised expressions. More precisely, for any synchronised expression $F = E\,\mathsf{sco}\,\rho$, we construct $\mathsf{itl}(F)$ directly from $\mathsf{itl}(E)$ by replacing each $\mathsf{flip}(t)$ by $\mathsf{flip}(\mathsf{trans}(t))$, and each $\mathsf{skipstbl}(V)$ by $\mathsf{skipstbl}(\bigcup \mathsf{trans}(V))$. The size of $\mathsf{itl}(F)$ would be similar as that of $\mathsf{itl}(\mathsf{stl}(F))$.

Theorem 4.3. (synchronised expression)
Let $F = E\,\mathsf{sco}\,\rho$ be a synchronised box expression. Then, $\mathit{sts}(\mathsf{itl}(F)) = \mathit{sts}(\mathsf{box}(F))$, where $\mathit{sts} = \mathsf{infsts}$ or $\mathit{sts} = \mathsf{finsts}$.

Proof:
Let us consider the streamlined expression $\mathsf{stl}(E\,\mathsf{sco}\,\rho) = E'\,\mathsf{sco}\,\rho'$, as defined in Section 2.6, and $\varphi = \mathsf{itl}(E'\,\mathsf{sco}\,\rho')$ as defined for streamlined expression. Then $\mathsf{itl}(E\,\mathsf{sco}\,\rho)$ and φ are equivalent after applying a consistent renaming of variables given by the bijection λ defined in Section 2.6. Now, we observe that if $\mathsf{trans}(a_\pi) = \{b^1_{W_1}, \ldots, b^m_{W_m}\}$ and $m \geq 2$, then the sub-formula $\mathsf{fs}(a_\pi \mid \varnothing)$ in $\mathsf{itl}(E)$ is transformed into $\mathsf{fs}(\mathsf{trans}(a_\pi) \mid \varnothing)$ within $\mathsf{itl}(F)$. This is equivalent to

$$\bigvee_{1 \leq i \leq m} \mathsf{fs}(b^i_{W_i} \mid b^1_{W_1}, \ldots, b^{i-1}_{W_{i-1}}, b^{i+1}_{W_{i+1}}, \ldots, b^m_{W_m}),$$

which in turn can be shown to be equivalent (after taking into account the correspondence given by λ) to $\mathsf{itl}(b^1_{W_1} \,\Box\, \ldots \,\Box\, b^m_{W_m})$. Hence, the result follows from Theorem 4.2 and Propositions 2.7 and 3.4. □

To conclude, we have demonstrated that it is possible to associate in a computationally efficient way a semantically equivalent ITL formula with any box expression considered in this paper.

[9] Such an assumption is usually made, e.g., by process algebras.

[10] Take, for every $n \geq 1$, $F_n = \underbrace{(\mathsf{stop} \parallel \mathsf{stop}) \,\Box\, \ldots \,\Box\, (\mathsf{stop} \parallel \mathsf{stop})}_{n \text{ times}})\,\mathsf{sco}\,\varnothing$. Then $|F_n| = 2 \cdot n$ and $|\mathsf{box}(F_n)| = 2^{n+1}$.

5. Examples

We will now present examples illustrating the translation of box expressions into logical formulas.

5.1. Synchronisation and restriction

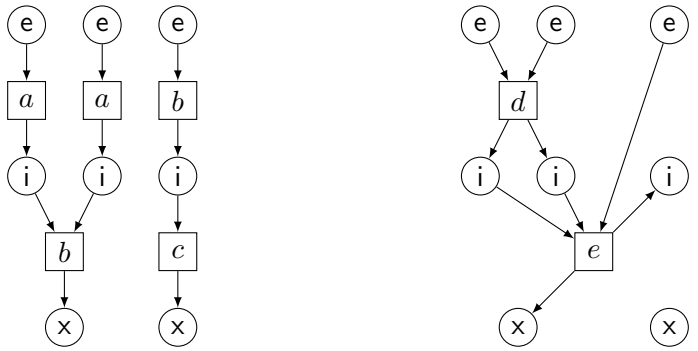


Figure 5. Boxes of $E = ((a \parallel a)\,;b) \parallel (b\,;c)$ and $F = E\,\mathsf{sco}\,\{aa \mapsto d, bb \mapsto e\}$.

To illustrate action restriction as well as internal and external synchronisations, we consider a streamlined expression $F = E\,\mathsf{sco}\,\rho$, where

$$E = ((a \parallel a)\,;b) \parallel (b\,;c) \quad \text{and} \quad \rho = \{aa \mapsto d, bb \mapsto e\}.$$

The corresponding boxes are shown in Figure 5. In the translation, we first derive:

$$\begin{array}{rcl} \mathsf{itl}(E) & = & \big((\mathsf{fs}(a_{\shortparallel_L\,;L\shortparallel_L} \mid b_{\shortparallel_L\,;R}) \wedge \mathsf{fs}(a_{\shortparallel_L\,;L\shortparallel_R} \mid b_{\shortparallel_L\,;R}))\,;\mathsf{fs}(b_{\shortparallel_L\,;R} \mid a_{\shortparallel_L\,;L\shortparallel_L}, a_{\shortparallel_L\,;L\shortparallel_R})\big) \\ & & \wedge\big(\mathsf{fs}(b_{\shortparallel_R\,;L} \mid c_{\shortparallel_R\,;R})\,;\mathsf{fs}(c_{\shortparallel_R\,;R} \mid b_{\shortparallel_R\,;L})\big). \end{array}$$

To prepare for applying scoping, we derive

$$\begin{array}{lllllll} \mathsf{trans}(a_{\shortparallel_L\,;L\shortparallel_L}) & = & \mathsf{trans}(a_{\shortparallel_L\,;L\shortparallel_R}) & = & \{d_{\shortparallel_L\,;L\shortparallel_L,\shortparallel_L\,;L\shortparallel_R}\} & = & \{\alpha\} \\ \mathsf{trans}(b_{\shortparallel_L\,;R}) & = & \mathsf{trans}(b_{\shortparallel_R\,;L}) & = & \{e_{\shortparallel_L\,;R,\shortparallel_R\,;L}\} & = & \{\beta\} \\ \mathsf{trans}(c_{\shortparallel_R\,;R}) & = & \varnothing & & & & \end{array}$$

which after simplifications leads to

$$\begin{array}{rcl} \mathsf{itl}(F) & = & ((\mathsf{fs}(\alpha \mid \beta) \wedge \mathsf{fs}(\alpha \mid \beta))\,;\mathsf{fs}(\beta \mid \alpha)) \;\wedge\; (\mathsf{fs}(\beta \mid \varnothing)\,;\mathsf{infstbl}(\beta)) \\ & = & (\mathsf{fs}(\alpha \mid \beta)\,;\mathsf{fs}(\beta \mid \alpha)) \;\wedge\; (\mathsf{fs}(\beta \mid \varnothing)\,;\mathsf{infstbl}(\beta)) \\ & = & \mathsf{fs}(\alpha \mid \beta)\,;\mathsf{fs}(\beta \mid \alpha)\,;\mathsf{infstbl}(\alpha, \beta). \end{array}$$

Thus $\mathsf{itl}(F)$ is satisfied over an interval provided that the latter can be split into three successive subintervals (the first two being finite and the third one infinite) overlapping on single states so that the

variables α and β are kept unchanged, except for one flipping of α within the first interval, and one flipping of β in the second interval (see (4) for the definitions of $\mathsf{fs}(\alpha \,|\, \beta)$, $\mathsf{fs}(\beta \,|\, \alpha)$, and $\mathsf{infstbl}(\alpha, \beta)$). Note that $\mathsf{finsts}(F) = \varnothing$ and $\mathsf{infsts}(F) = \varnothing^\omega \cup \varnothing^*{\cdot}\{\{\alpha\}\}{\cdot}\varnothing^\omega \cup \varnothing^*{\cdot}\{\{\alpha\}\}{\cdot}\varnothing^*{\cdot}\{\{\beta\}\}{\cdot}\varnothing^\omega$.

5.2. Parallel composition and choice

To illustrate choice and parallel composition, we consider a streamlined expression $F = E \,\mathsf{sco}\, \rho$, where

$$E = (a \,\square\, b) \parallel b \quad \text{and} \quad \rho = \{a \mapsto a, bb \mapsto b\}.$$

The corresponding boxes are shown in Figure 6. In the translation, we first derive

$$\mathsf{itl}(E) \;=\; (\mathsf{fs}(a_{\shortparallel_L \,\square_L} \,|\, b_{\shortparallel_L \,\square_R}) \vee \mathsf{fs}(b_{\shortparallel_L \,\square_R} \,|\, a_{\shortparallel_L \,\square_L})) \;\wedge\; \mathsf{fs}(b_{\shortparallel_R} \,|\, \varnothing).$$

To prepare for applying scoping, we derive:

$$\begin{array}{lllll} \mathsf{trans}(a_{\shortparallel_L \,\square_L}) & & = & \{a_{\shortparallel_L \,\square_L}\} & = \{\alpha\} \\ \mathsf{trans}(b_{\shortparallel_L \,\square_R}) \;=\; \mathsf{trans}(b_{\shortparallel_R}) & & = & \{b_{\shortparallel_L \,\square_R, \shortparallel_R}\} & = \{\beta\} \end{array}$$

which (after eliminating true in conjunctions) leads to

$$\mathsf{itl}(F) \;=\; (\mathsf{fs}(\alpha \,|\, \beta) \vee \mathsf{fs}(\beta \,|\, \alpha)) \wedge \mathsf{fs}(\beta \,|\, \varnothing) \;\equiv\; (\mathsf{fs}(\alpha \,|\, \beta)\,;\mathsf{infstbl}(\alpha, \beta)) \vee \mathsf{fs}(\beta \,|\, \alpha).$$

Hence $\mathsf{finsts}(F) = \varnothing^*{\cdot}\{\{\beta\}\}{\cdot}\varnothing^*$ and $\mathsf{infsts}(F) = \varnothing^\omega \cup \varnothing^*{\cdot}\{\{\alpha\}, \{\beta\}\}{\cdot}\varnothing^\omega$.

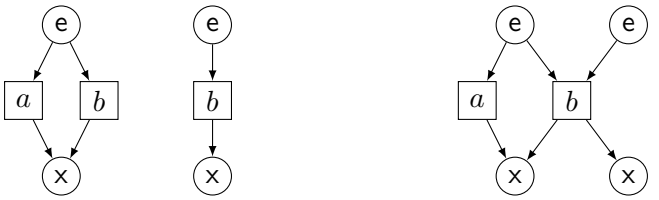


Figure 6. Boxes of $E = (a \,\square\, b) \parallel b$ and $F = E \,\mathsf{sco}\, \{a \mapsto a, bb \mapsto b\}$.

5.3. Iteration and scoping

To illustrate synchronisation inside iteration, we consider a non-streamlined expression $F = E \,\mathsf{sco}\, \rho$, where

$$E = [\![a \circledast (b \parallel c) \circledast d]\!] \quad \text{and} \quad \rho = \{a \mapsto a, bc \mapsto a, c \mapsto e\}.$$

The boxes corresponding to E and F are shown in Figure 4. We first derive

$$\mathsf{itl}(E) \;=\; \mathsf{fs}(a_{\circledast_L} \,|\, b_{\circledast_{M^{\shortparallel_L}}}, c_{\circledast_{M^{\shortparallel_R}}}, d_{\circledast_R})\,; \begin{pmatrix} \mathsf{fs}(b_{\circledast_{M^{\shortparallel_L}}} \,|\, a_{\circledast_L}, d_{\circledast_R}) \\ \wedge \\ \mathsf{fs}(c_{\circledast_{M^{\shortparallel_R}}} \,|\, a_{\circledast_L}, d_{\circledast_R}) \end{pmatrix}^*\,; \mathsf{fs}(d_{\circledast_R} \,|\, a_{\circledast_L}, b_{\circledast_{M^{\shortparallel_L}}}, c_{\circledast_{M^{\shortparallel_R}}}).$$

To prepare for applying scoping, we derive

$$\begin{array}{lcll} \mathsf{trans}(a_{\circledast_L}) & = & \{a_{\circledast_L}\} & = \{\gamma\} \\ \mathsf{trans}(d_{\circledast_R}) & = & \varnothing & \\ \mathsf{trans}(b_{\circledast_{M\shortparallel L}}) & = & \{a_{\circledast_{M\shortparallel L},\circledast_{M\shortparallel R}}\} & = \{\alpha\} \\ \mathsf{trans}(c_{\circledast_{M\shortparallel R}}) & = & \{a_{\circledast_{M\shortparallel L},\circledast_{M\shortparallel R}}, e_{\circledast_{M\shortparallel R}}\} & = \{\alpha,\zeta\} \end{array}$$

which leads to

$$\begin{array}{rcl} \mathsf{itl}(F) & = & \mathsf{fs}(\gamma \,|\, \alpha,\zeta)\,;\ \big(\mathsf{fs}(\alpha \,|\, \gamma) \wedge \mathsf{fs}(\alpha,\zeta \,|\, \gamma)\big)^*\,;\mathsf{infstbl}(\gamma,\alpha,\zeta) \\ & \equiv & \mathsf{fs}(\gamma \,|\, \alpha,\zeta)\,;\big((\mathsf{fs}(\alpha \,|\, \gamma) \wedge \mathsf{fs}(\alpha \,|\, \gamma,\zeta)) \vee (\mathsf{fs}(\alpha \,|\, \gamma) \wedge \mathsf{fs}(\zeta \,|\, \gamma,\alpha))\big)^*\,;\mathsf{infstbl}(\gamma,\alpha,\zeta) \\ & \equiv & \mathsf{fs}(\gamma \,|\, \alpha,\zeta)\,;\big(\mathsf{fs}(\alpha \,|\, \gamma,\zeta) \vee (\mathsf{fs}(\zeta \,|\, \gamma,\alpha)\,;\mathsf{infstbl}(\gamma,\alpha,\zeta))\big)^*\,;\mathsf{infstbl}(\gamma,\alpha,\zeta). \end{array}$$

Hence $\mathsf{infsts}(F) = \varnothing^\omega \cup \varnothing^*\cdot\{\{\gamma\}\}\cdot\big\{\varnothing^\omega \cup (\varnothing^*\cdot\{\{\alpha\}\})^*\cdot\varnothing^\omega \cup (\varnothing^*\cdot\{\{\alpha\}\})^*\cdot\varnothing^*\cdot\{\{\zeta\}\}\cdot\varnothing^\omega\big\}$ and $\mathsf{finsts}(F) = \varnothing$.

5.4. Producer/consumers system

Consider BA expressions G_n ($n \geq 1$), modelling a producer P_n working in parallel with n consumer processes $C_1, \ldots, C_n$. After starting up using action a, the producer repeatedly performs a local action b followed by a parallel execution of communication actions $c_1, \ldots, c_n$, each cycle being finished by the execution of action d, and then a new cycle is started by executing action e. A consumer process C_i, after a start up action a_i, also executes an indefinite loop executing action c_i and ending each cycle with f_i. The definitions of these expressions are as follows:

$$\begin{array}{rcll} C_i & = & [\![a_i \circledast c_i\,;f_i \circledast \mathsf{stop}]\!] & (i = 1,\ldots,n) \\ P_n & = & [\![a \circledast b\,;(c_1 \shortparallel \ldots \shortparallel c_n)\,;d\,;e \circledast \mathsf{stop}]\!] & \\ PC_n & = & (C_1 \shortparallel \ldots \shortparallel C_n \shortparallel P_n)\,\mathsf{sco}\,\rho & \end{array}$$

where

$$\rho = \left\{ \begin{array}{cccccc} c_1c_1 \mapsto \tau_1 & \ldots & c_nc_n \mapsto \tau_n & a \mapsto a & & \\ a_1 \mapsto a_1 & \ldots & a_n \mapsto a_n & d \mapsto d & b \mapsto b & \\ f_1 \mapsto f_1 & \ldots & f_n \mapsto f_n & e \mapsto e & & \end{array} \right\}$$

Figure 7 depicts the boxes of C_i and P_n. Moreover, Figure 8 shows the box corresponding to PC_2 together with the initial marking. The corresponding ITL formula is shown below (for brevity, we only indicate variables which flip their values, but even then we omit the subscripts, and show only the labels of transitions corresponding to these variables):

$$\begin{array}{rcl} \mathsf{itl}(PC_n) & = & \big(\mathsf{fs}(a_1 \,|\,)\,;(\mathsf{fs}(\tau_1 \,|\,)\,;\mathsf{fs}(f_1 \,|\,))^*\,;\mathsf{infstbl}()\big) \ \wedge \cdots \wedge \\ & & \big(\mathsf{fs}(a_n \,|\,)\,;(\mathsf{fs}(\tau_n \,|\,)\,;\mathsf{fs}(f_n \,|\,))^*\,;\mathsf{infstbl}()\big) \ \wedge \\ & & \big(\mathsf{fs}(a \,|\,)\,;\big(\mathsf{fs}(b \,|\,)\,;(\mathsf{fs}(\tau_1 \,|\,) \wedge \cdots \wedge \mathsf{fs}(\tau_n \,|\,))\,;\mathsf{fs}(d \,|\,)\,;\mathsf{fs}(e \,|\,)\big)^*\,;\mathsf{infstbl}()\big). \end{array}$$

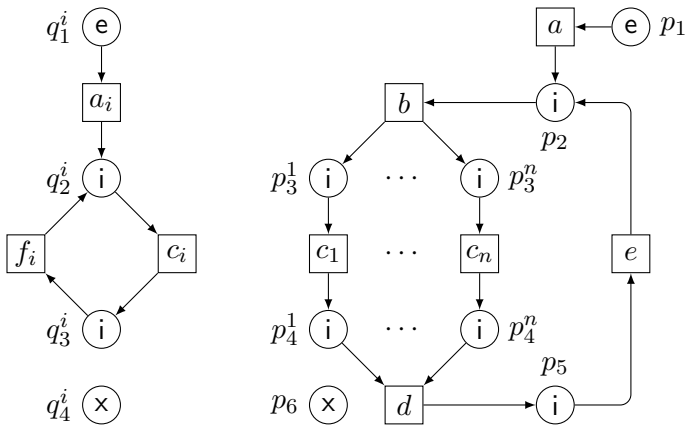


Figure 7. The boxes of the consumers C_i and the producer P_n.

6. Conclusions

In the past, logics have been mainly used for expressing correctness properties of systems specified using Petri nets [36]. When it comes to the intrinsic semantic relationship between logics and Petri nets, we feel that the work on the connections between linear logic [37] and Place Transition nets has been the closest one. However, the main concern there was the handling of multiple token occurrences in net places whereas here nets can hold at most two tokens in a single place. Another way in which logics and Petri nets are related is reported in [38], which provided a characterisation of Petri net languages in terms of second-order logical formulas.

The results presented in this paper demonstrate that one can define a translation from box expressions to ITL with equivalent behaviour. We also show that the complexity of the proposed translation compares favourably with the complexity of the translation from box expressions to boxes.

It is therefore important to further investigate the extent to which the established connection between BA and ITL could be generalised and exploited. In particular, we plan to investigate what is the subset of ITL which can be translated into BA. We do not expect that such a translation will be easy for the full ITL for at least two reasons. The first is that the projection operator prj of ITL does not have an equivalent in BA and so incorporating it would call for new constructs at the syntactical and semantical levels. The second reason is that ITL allows one to specify actions happening simultaneously, e.g., , as in $\mathsf{flip}(v) \wedge \mathsf{flip}(w)$. This is akin to executing the step $\{v, w\}$ without being able to execute $\{v\}\{w\}$ or $\{w\}\{v\}$, i.e., v and w must be executed synchronously and without delay. Such a behaviour cannot be reproduced in the standard Petri nets (and also boxes) as they are inherently asynchronous models. The required effect could perhaps be achieved by using the maximally concurrent execution rule, but this would fundamentally change the net model. In fact, we feel that in logical formulas translatable to box expressions all actions should be fully asynchronous, using $\mathsf{fs}(v \mid \varnothing)$ rather than v in ITL syntax. However, more investigation needs to be conducted in order, in particular, to generate a suitable scoping set from a give ITL formula.

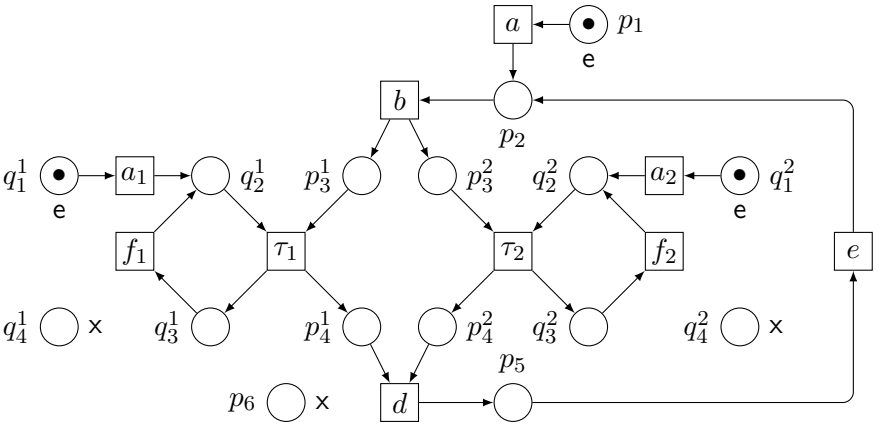


Figure 8. Initially marked box of PC_2.

A long-term goal is the development of a hybrid verification methodology combining ITL and BA techniques. For example, sequential algorithms and infinite data structures could be treated by ITL techniques [39, 40, 41, 30], while intensive parallel or communicating aspects of systems could be treated by net unfoldings [42, 43] or other Petri net techniques [44].

Acknowledgement

This research was supported by the 973 Program Grant 2010CB328102, NSFC Grant 61133001, ANR SYNBIOTIC and EPSRC UNCOVER project.

References

[1] Goranko V, Montanari A, Sciavicco G. A Road Map of Interval Temporal Logics and Duration Calculi. *Journal of Applied Non-Classical Logics*, 2004. **14**(1-2):9–54. doi:10.3166/jancl.14.9-54. URL `https://doi.org/10.3166/jancl.14.9-54`.

[2] Emerson EA. Temporal and Modal Logic. In: Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B), pp. 995–1072. MIT Press Cambridge, 1990.

[3] Manna Z, Pnueli A. Verification of Concurrent Programs: Temporal Proof Principles. In: Kozen D (ed.), Logics of Programs, Workshop, Yorktown Heights, New York, USA, May 1981, volume 131 of *Lecture Notes in Computer Science*. Springer. ISBN 3-540-11212-X, 1981 pp. 200–252. doi: 10.1007/BFb0025785. URL `https://doi.org/10.1007/BFb0025785`.

[4] Moszkowski BC, Guelev DP, Leucker M. Guest editors' preface to special issue on interval temporal logics. *Ann. Math. Artif. Intell.*, 2014. **71**(1-3):1–9. doi:10.1007/s10472-014-9417-7. URL `https://doi.org/10.1007/s10472-014-9417-7`.

[5] Desel J, Juhás G. "What Is a Petri Net?". In: Ehrig H, Juhás G, Padberg J, Rozenberg G (eds.), Unifying Petri Nets, Advances in Petri Nets, volume 2128 of *Lecture Notes in Computer*

Science. Springer. ISBN 3-540-43067-9, 2001 pp. 1–25. doi:10.1007/3-540-45541-8_1. URL https://doi.org/10.1007/3-540-45541-8_1.

[6] Peterka G, Murata T. Proof Procedure and Answer Extraction in Petri Net Model of Logic Programs. *IEEE Trans. Software Eng.*, 1989. **15**(2):209–217. doi:10.1109/32.21746. URL https://doi.org/10.1109/32.21746.

[7] Suárez MS, Teruel E, Colom JM. Linear Algebraic and Linear Programming Techniques for the Analysis of Place or Transition Net Systems. In: Reisig W, Rozenberg G (eds.), Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, the volumes are based on the Advanced Course on Petri Nets, held in Dagstuhl, September 1996, volume 1491 of *Lecture Notes in Computer Science*. Springer. ISBN 3-540-65306-6, 1996 pp. 309–373. doi:10.1007/3-540-65306-6_19. URL https://doi.org/10.1007/3-540-65306-6_19.

[8] McMillan KL. A Technique of State Space Search Based on Unfolding. *Formal Methods in System Design*, 1995. **6**(1):45–65. doi:10.1007/BF01384314. URL https://doi.org/10.1007/BF01384314.

[9] Valmari A. Stubborn sets for reduced state space generation. In: Rozenberg G (ed.), Advances in Petri Nets 1990 [10th International Conference on Applications and Theory of Petri Nets, Bonn, Germany, June 1989, Proceedings], volume 483 of *Lecture Notes in Computer Science*. Springer. ISBN 3-540-53863-1, 1989 pp. 491–515. doi:10.1007/3-540-53863-1_36. URL https://doi.org/10.1007/3-540-53863-1_36.

[10] Duan Z, Klaudel H, Koutny M. ITL semantics of composite Petri nets. *J. Log. Algebr. Program.*, 2013. **82**(2):95–110. doi:10.1016/j.jlap.2012.12.001. URL https://doi.org/10.1016/j.jlap.2012.12.001.

[11] Moszkowski BC. Compositional reasoning about projected and infinite time. In: 1st IEEE International Conference on Engineering of Complex Computer Systems (ICECCS '95), November 6-10, 1995, Fort Lauderdale, Florida, USA. IEEE Computer Society. ISBN 0-8186-7123-8, 1995 pp. 238–245. doi:10.1109/ICECCS.1995.479336. URL https://doi.org/10.1109/ICECCS.1995.479336.

[12] and Zohar Manna. Reasoning in Interval Temporal Logic. In: Clarke EM, Kozen D (eds.), Logics of Programs, Workshop, Carnegie Mellon University, Pittsburgh, PA, USA, June 6-8, 1983, Proceedings, volume 164 of *Lecture Notes in Computer Science*. Springer. ISBN 3-540-12896-4, 1983 pp. 371–382. doi:10.1007/3-540-12896-4_374. URL https://doi.org/10.1007/3-540-12896-4_374.

[13] Best E, Devillers R, Koutny M. Petri Net Algebra. Monographs in Theoretical Computer Science. Springer, 2001.

[14] Best E, Devillers RR, Hall JG. The box calculus: a new causal algebra with multi-label communication. In: Rozenberg G (ed.), Advances in Petri Nets 1992, The DEMON Project, volume 609 of *Lecture Notes in Computer Science*, pp. 21–69. Springer. ISBN 3-540-55610-9, 1992. doi:10.1007/3-540-55610-9_167. URL https://doi.org/10.1007/3-540-55610-9_167.

[15] Milner R. A Calculus of Communicating Systems. Springer, 1980.

[16] CARHoare. Communicating Sequential Processes. Prentice-Hall, 1985.

[17] Best E, Fraczak W, Hopkins RP, Klaudel H, Pelz E. M-Nets: An Algebra of High-Level Petri Nets, with an Application to the Semantics of Concurrent Programming Languages. *Acta Inf.*, 1998. **35**(10):813–857. doi:10.1007/s002360050144. URL https://doi.org/10.1007/s002360050144.

[18] Macià H, Ruiz VV, Cuartero F, de Frutos-Escrig D. A congruence relation for sPBC. *Formal Methods in System Design*, 2008. **32**(2):85–128. doi:10.1007/s10703-007-0045-2. URL https://doi.org/10.1007/s10703-007-0045-2.

[19] Goranko V, Montanari A. Foreword to Special Issue on Interval Temporal Logics and Duration Calculi. *Journal of Applied Non-Classical Logics*, 2004. **14**(1-2):7–8.

[20] Bäumler S, Schellhorn G, Tofan B, Reif W. Proving linearizability with temporal logic. *Formal Asp. Comput.*, 2011. **23**(1):91–112. doi:10.1007/s00165-009-0130-y. URL https://doi.org/10.1007/s00165-009-0130-y.

[21] Halpern JY, Shoham Y. A Propositional Modal Logic of Time Intervals. *J. ACM*, 1991. **38**(4):935–962. doi:10.1145/115234.115351. URL https://doi.org/10.1145/115234.115351.

[22] Venema Y. A Modal Logic for Chopping Intervals. *J. Log. Comput.*, 1991. **1**(4):453–476. doi:10.1093/logcom/1.4.453. URL https://doi.org/10.1093/logcom/1.4.453.

[23] Allen JF. Maintaining Knowledge about Temporal Intervals. *Commun. ACM*, 1983. **26**(11):832–843. doi:10.1145/182.358434. URL http://doi.acm.org/10.1145/182.358434.

[24] Bozzelli L, Molinari A, Montanari A, Peron A, Sala P. Interval vs. Point Temporal Logic Model Checking: An Expressiveness Comparison. *ACM Trans. Comput. Log.*, 2019. **20**(1):4:1–4:31. doi:10.1145/3281028. URL https://doi.org/10.1145/3281028.

[25] Lomuscio A, Michaliszyn J. An Epistemic Halpern-Shoham Logic. In: Rossi F (ed.), IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013. IJCAI/AAAI. ISBN 978-1-57735-633-2, 2013 pp. 1010–1016. URL http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6632.

[26] Lomuscio AR, Michaliszyn J. Decidability of model checking multi-agent systems against a class of EHS specifications. In: Schaub T, Friedrich G, O'Sullivan B (eds.), ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014), volume 263 of *Frontiers in Artificial Intelligence and Applications*. IOS Press. ISBN 978-1-61499-418-3, 2014 pp. 543–548. doi:10.3233/978-1-61499-419-0-543. URL https://doi.org/10.3233/978-1-61499-419-0-543.

[27] Montanari A, Murano A, Perelli G, Peron A. Checking Interval Properties of Computations. In: Cesta A, Combi C, Laroussinie F (eds.), 21st International Symposium on Temporal Representation and Reasoning, TIME 2014, Verona, Italy, September 8-10, 2014. IEEE Computer Society. ISBN 978-1-4799-4228-2, 2014 pp. 59–68. doi:10.1109/TIME.2014.24. URL https://doi.org/10.1109/TIME.2014.24.

[28] Molinari A, Montanari A, Murano A, Perelli G, Peron A. Checking interval properties of computations. *Acta Inf.*, 2016. **53**(6-8):587–619. doi:10.1007/s00236-015-0250-1. URL https://doi.org/10.1007/s00236-015-0250-1.

[29] http://www.antonio-cau.co.uk/ITL/.

[30] Moszkowski BC. Executing Temporal Logic Programs. Cambridge University Press, 1986.

[31] IEEE: Standard for the Functional Verification Language e, Standard 1647-2011. ANSI/IEEE, New York, 2011.

[32] Chaochen Z, Hoare CAR, Ravn AP. A Calculus of Durations. *Inf. Process. Lett.*, 1991. **40**(5):269–276. doi:10.1016/0020-0190(91)90122-X. URL https://doi.org/10.1016/0020-0190(91)90122-X.

[33] Linker S, Hilscher M. Proof Theory of a Multi-Lane Spatial Logic. *Logical Methods in Computer Science*, 2015. **11**(3). doi:10.2168/LMCS-11(3:4)2015. URL https://doi.org/10.2168/LMCS-11(3:4)2015.

[34] Klaudel H, Koutny M, Moszkowski BC. From Petri Nets with Shared Variables to ITL. In: Desel J, Yakovlev A (eds.), 16th International Conference on Application of Concurrency to System Design, ACSD 2016, Torun, Poland, June 19-24, 2016. IEEE Computer Society. ISBN 978-1-5090-2589-3, 2016 pp. 11–18. doi:10.1109/ACSD.2016.12. URL https://doi.org/10.1109/ACSD.2016.12.

[35] Klaudel H, Koutny M, Duan Z. Interval Temporal Logic Semantics of Box Algebra. In: Dediu A, Martín-Vide C, Sierra-Rodríguez JL, Truthe B (eds.), Language and Automata Theory and Applications - 8th International Conference, LATA 2014, Madrid, Spain, March 10-14, 2014. Proceedings, volume 8370 of *Lecture Notes in Computer Science*. Springer. ISBN 978-3-319-04920-5, 2014 pp. 441–452. doi:10.1007/978-3-319-04921-2_36. URL https://doi.org/10.1007/978-3-319-04921-2_36.

[36] Esparza J, Nielsen M. Decidability Issues for Petri Nets - a survey. *Elektronische Informationsverarbeitung und Kybernetik*, 1994. **30**(3):143–160.

[37] Girard J. Linear Logic. *Theor. Comput. Sci.*, 1987. **50**:1–102. doi:10.1016/0304-3975(87)90045-4. URL https://doi.org/10.1016/0304-3975(87)90045-4.

[38] Parigot M, Pelz E. A Logical Approach of Petri Net Languages. *Theor. Comput. Sci.*, 1985. **39**:155–169. doi:10.1016/0304-3975(85)90136-7. URL https://doi.org/10.1016/0304-3975(85)90136-7.

[39] Cau A, Janicke H, Moszkowski BC. Verification and enforcement of access control policies. *Formal Methods in System Design*, 2013. **43**(3):450–492. doi:10.1007/s10703-013-0187-3. URL https://doi.org/10.1007/s10703-013-0187-3.

[40] Cau A, Zedan H. Refining Interval Temporal Logic Specifications. In: Bertran M, Rus T (eds.), Transformation-Based Reactive Systems Development, 4th International AMAST Workshop on Real-Time Systems and Concurrent and Distributed Software, ARTS'97, Palma, Mallorca, Spain, May 21-23, 1997, Proceedings, volume 1231 of *Lecture Notes in Computer Science*. Springer. ISBN 3-540-63010-4, 1997 pp. 79–94. doi:10.1007/3-540-63010-4_6. URL https://doi.org/10.1007/3-540-63010-4_6.

[41] Janicke H, Cau A, Siewe F, Zedan H. Dynamic Access Control Policies: Specification and Verification. *Comput. J.*, 2013. **56**(4):440–463. doi:10.1093/comjnl/bxs102. URL https://doi.org/10.1093/comjnl/bxs102.

[42] Esparza J, Römer S, Vogler W. An Improvement of McMillan's Unfolding Algorithm. In: Margaria T, Steffen B (eds.), Tools and Algorithms for Construction and Analysis of Systems, Second International Workshop, TACAS '96, Passau, Germany, March 27-29, 1996, Proceedings, volume 1055 of *Lecture Notes in Computer Science*. Springer. ISBN 3-540-61042-1, 1996 pp. 87–106. doi:10.1007/3-540-61042-1_40. URL https://doi.org/10.1007/3-540-61042-1_40.

[43] Khomenko V, Koutny M. Towards an Efficient Algorithm for Unfolding Petri Nets. In: Larsen KG, Nielsen M (eds.), CONCUR 2001 - Concurrency Theory, 12th International Conference, Aalborg, Denmark, August 20-25, 2001, Proceedings, volume 2154 of *Lecture Notes in Computer Science*. Springer. ISBN 3-540-42497-0, 2001 pp. 366–380. doi:10.1007/3-540-44685-0_25. URL https://doi.org/10.1007/3-540-44685-0_25.

[44] http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/.